



**Filipe Machado
da Cunha**

Simulação de redes de comunicações em NS



**Filipe Machado
da Cunha**

Simulação de redes de comunicações em NS

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Professor Doutor Rui Jorge Tomaz Valadas, Professor Associado do Departamento de Engenharia Electrónica e Telecomunicações da Universidade de Aveiro, e co-orientação da Professora Doutora Susana Isabel Barreto de Miranda Sargento, Professora Auxiliar Convidada do Departamento de Engenharia Electrónica e Telecomunicações da Universidade de Aveiro.

Dedico este trabalho à minha esposa pelo incansável apoio e ao meu filho.

o júri

presidente

Prof. Dr. Joaquim Arnaldo Carvalho Martins
Professor Catedrático da Universidade de Aveiro

Prof. Dr. Rui Jorge Morais Tomaz Valadas
Professor Associado da Universidade de Aveiro

Prof. Dr. Manuel Alberto Pereira Ricardo
Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto

Prof. Dr. Susana Isabel Barreto de Miranda Sargento
Professor Auxiliar Convidada da Universidade de Aveiro

agradecimentos

Foi um enorme prazer ter desenvolvido este trabalho sob a orientação dos professores Rui Valadas e Susana Sargento.

Ao Professor Rui Valadas por desde logo ter acedido orientar esta dissertação e, principalmente, por me ter dado a oportunidade de trabalhar numa área tão estimulante.

À Professora Susana Sargento por ter sido incansável na melhoria do trabalho e no alcance do rigor.

A ambos por todo o apoio e disponibilidade demonstrados ao longo deste tempo, o meu muito obrigado.

Agradeço aos meus amigos, Elza Silva e Rogério Azevedo que, enquanto partilhávamos conhecimentos e angústias, apoiavam-me no desenvolvimento da dissertação.

À minha família: Pai, Mãe e irmãos, obrigado por estarem sempre presentes e sempre me ter apoiado.

A todos os meus amigos, que sabem que não posso fazer distinções particulares por serem todos muito importantes. São eles que estão presentes nos bons e nos maus momentos.

Ao meu filho Dinis, que não existia quando esta aventura começou e que se transformou na minha fonte de inspiração.

Finalmente, à minha mulher Carmo, porque sem ela eu não teria conseguido. A ela devo-lhe muito, mesmo muito.

A todos muito obrigado.

palavras-chave

Redes de Comunicações, Simulação, Sistema Fila de espera, RED, Algoritmo de escalonamento, TCP, QoS.

resumo

As redes de comunicações têm assumido um papel cada vez mais importante no quotidiano das pessoas, empresas e instituições. O constante desenvolvimento das redes de comunicações e a globalização da informação permitiu assistir, em menos de meia década, ao aumento dos débitos de transferência da informação no acesso à Internet através da linha do assinante, de 56 *Kbit/s* para 16 *Mbit/s*.

As redes de comunicações são cada vez mais sofisticadas e complexas e os serviços suportados estão em constante renovação e crescimento.

A engenharia de Tráfego assume uma importância fundamental neste cenário complexo em constante evolução, fornecendo metodologias para dimensionar os recursos e parametrizar os mecanismos de controlo das redes, de forma eficiente.

Uma das ferramentas utilizadas pela engenharia de Tráfego é a simulação. A simulação permite levar a cabo estudos de desempenho das redes de comunicações com grande detalhe, em geral superior ao dos modelos de análise teórica. A simulação possibilita (i) avaliar e completar modelos de análise teóricos, (ii) escolher entre as diferentes alternativas disponíveis para implementar os vários mecanismos de controlo de rede e determinar os valores mais adequados dos seus parâmetros, (iii) dimensionar os recursos das redes de forma eficiente e (iv) levar a cabo estudos de evolução das redes com base em estimativas de crescimento do tráfego.

Esta dissertação tem como objectivo estudar um conjunto de problemas de desempenho em redes de comunicações com o auxílio do simulador de redes NS-2 (*Network Simulator*).

Inicialmente é efectuado um estudo de simulação dos principais sistemas de filas de espera, comparando-se os resultados teóricos com os resultados obtidos por simulação. De seguida, são estudados por simulação diversos mecanismos de controlo de redes, nomeadamente os algoritmos de escalonamento, as técnicas de descarte de pacotes, os mecanismos de controlo de congestionamento e de policiamento de fluxo. No final, as arquitecturas de Qualidade de Serviço são analisadas.

Como parte deste trabalho, foram efectuados acrescentos ao NS-2, nomeadamente a criação de dois novos objectos que permitem gerar tráfego de pacotes com chegadas de *Poisson* e comprimentos exponencialmente distribuídos e que permitem extrair todas as medidas de desempenho de um sistema de fila de espera.

keywords

Communications Network, Simulation, Queuing system, Scheduling algorithms, RED, TCP, QoS.

abstract

The role of the communications networks is being increasingly important in the daily life of the persons, companies and institutions. The constant development of the communications network and the globalization of the information allowed us to attend, in less than one half of decade, to the increase of the information transmission rate in the Internet access through the subscriber lines, from 56 *Kbit/s* to 16 *Mbit/s*.

The communications networks are increasingly sophisticated and complex and the supported service are constantly renewing and growing.

The Traffic engineering is of fundamental importance in this complex environment in constant evolution, supplying methodologies for an efficient network dimensioning and choice of the correct parameters of the network control mechanisms.

One of the tools usually used by the Traffic engineering is the simulation. The simulation allows the accomplishment of performance studies of the communications networks with significant detail, in general larger than the one provided by analytical models. The simulation enables to (i) evaluate and enhance theoretical models, (ii) choose between different available control network mechanisms and evaluate the adequate values of its parameters, (iii) dimension the network resources in an efficient way and (iv) study the network evolution with information on the traffic growth estimation.

The objective of this master thesis is to study a set of network communication performance issues with the aid of the simulator networks NS-2 (Network Simulator).

Initially it is performed a simulation study of the main queuing systems, comparing the theoretical results with the ones obtained by simulation. Next, several control mechanisms of the networks are studied by simulation, such as the scheduling algorithms, the packet discards mechanisms, the congestion control mechanisms and traffic regulation mechanisms. At the end, the Quality of Service architectures are studied.

As part of this work, add-ons to the NS-2 simulator were performed, namely, the creation of two new objects that generate packet traffic with *Poisson* inter-arrival time and exponentially distributed packets length, and that enable the extraction of all the performance metrics in a queuing system.

Índice Geral

1.	Introdução	3
1.1	MOTIVAÇÃO	3
1.2	OBJECTIVOS DA DISSERTAÇÃO	5
1.3	ORGANIZAÇÃO DA DISSERTAÇÃO	5
2.	O simulador, NS-2.....	7
2.1	ESTRUTURA DO NS.....	10
2.2	CENÁRIO DE SIMULAÇÃO	11
2.2.1	Nó	12
2.2.2	Ligação.....	12
2.2.3	Agente.....	14
2.2.4	Gerador de Tráfego e Aplicação	15
2.2.5	Agendar os eventos discretos	16
2.3	EXTRAIR RESULTADOS DA SIMULAÇÃO.....	16
2.3.1	Leitura das variáveis dos objectos	17
2.3.2	Registo de actividade da rede (Trace).....	18
2.3.3	Monitorização da fila de espera.....	20
2.3.4	Ferramentas de Visualização.....	23
2.4	CONCLUSÕES.....	25
3.	Sistemas de filas de espera.....	27
3.1	M/M/1	28
3.2	M/G/1.....	29
3.3	SISTEMA M/G/1 COM PRIORIDADES.....	31
3.4	APROXIMAÇÃO DE KLEINROCK.....	32
3.5	OBJECTOS ADICIONADOS AO NS	34
3.5.1	Nova fonte de tráfego Exp_Exp	34
3.5.2	Novo agente para estimar medidas de desempenho do sistema	36
3.6	SIMULAÇÕES.....	39
3.6.1	Sistema M/M/1	39
3.6.2	Sistema M/G/1.....	40
3.6.3	Sistema M/G/1 com prioridades	42
3.6.4	Aproximação de Kleinrock.....	43
3.7	CONCLUSÕES.....	45
4.	Algoritmos de escalonamento	47
4.1	FIFO – FIRST IN FIRST OUT	48
4.2	FQ – FAIR QUEUEING	48
4.3	SFQ – STOCHASTIC FAIR QUEUEING.....	50
4.4	DRR – DEFICIT ROUND ROBIN	52
4.5	CBQ – CLASS-BASED QUEUEING	53
4.6	SIMULAÇÕES.....	55
4.6.1	Caso 1, sentido $5 \rightarrow 6$	56
4.6.2	Caso 2, sentido $6 \rightarrow 5$	60
4.6.3	CBQ.....	63
4.7	CONCLUSÕES.....	66
5.	Técnicas de descarte	69
5.1	POSIÇÃO DO PACOTE A DESCARTAR E CONDIÇÃO DE DESCARTE.....	70
5.2	RED – RANDOM EARLY DETECTION	70
5.3	SIMULAÇÕES.....	73
5.3.1	DropTail Vs DropFront - Posição do pacote a descartar.....	74
5.3.2	RED - Parâmetros da fila de espera.....	77
5.3.3	DropTail Vs RED - Condição para o descarte de pacotes	78
5.3.4	DropTail Vs RED - Sincronização global.....	83
5.4	CONCLUSÕES.....	86

6.	Controlo de congestionamento	89
6.1	INTRODUÇÃO	90
6.2	RELÓGIO DE RETRANSMISSÃO - TIMEOUT	92
6.2.1	<i>Algoritmo Original</i>	92
6.2.2	<i>Algoritmo de Karn</i>	93
6.2.3	<i>Algoritmo de Jacobson</i>	94
6.3	MECANISMOS DE CONTROLO DE CONGESTIONAMENTO	94
6.3.1	<i>Sliding Window</i>	95
6.3.2	<i>Slow Start</i>	97
6.3.3	<i>Congestion Avoidance</i>	98
6.3.4	<i>Fast Retransmit</i>	100
6.3.5	<i>Fast Recovery</i>	101
6.3.6	<i>Alterações efectuadas pelo TCP Vegas</i>	103
6.4	SIMULAÇÕES	105
6.4.1	<i>Comparação entre as versões de TCP</i>	106
6.4.2	<i>TCP Reno Vs TCP Vegas</i>	110
6.5	CONCLUSÕES	126
7.	Mecanismos de policiamento	129
7.1	TOKEN BUCKET	129
7.2	SINGLE RATE THREE COLOR MARKER	130
7.3	TWO RATE THREE COLOR MARKER	132
7.4	TIME SLIDING WINDOW THREE COLOUR MARKER	133
7.5	SIMULAÇÕES	135
7.6	CONCLUSÕES	138
8.	QoS – Qualidade de Serviço	141
8.1	ARQUITECTURAS QoS	143
8.1.1	<i>Integrated Services - IntServ</i>	143
8.1.2	<i>Differentiated Services - DiffServ</i>	145
8.2	IMPLEMENTAÇÃO NO NS	149
8.3	SIMULAÇÕES	150
8.3.1	<i>Serviço Best-effort</i>	151
8.3.2	<i>Serviço DiffServ – PHB AF</i>	153
8.3.3	<i>Serviço DiffServ – PHB EF</i>	159
8.4	CONCLUSÕES	161
9.	Conclusões	165
9.1	DIRECTIVAS PARA TRABALHO FUTURO	168
10.	Referências	169

1. Introdução

1.1 Motivação

As redes de comunicações têm assumido um papel cada vez mais importante no quotidiano das pessoas, das empresas e das instituições de ensino. Com a massificação do acesso e da utilização da Internet, um número de lares portugueses já está ligado ao *World Wide Web* (WWW) [21].

As redes de comunicações estão em constante desenvolvimento. O simples par de cobre, que até há pouco tempo era utilizado para ligar a nossa casa à rede telefónica e nos permitia alternadamente efectuar uma chamada telefónica ou aceder à Internet a uma velocidade de 56 *Kbit/s*, com o surgimento do ADSL (*Asymmetric Digital Subscriber Line*), permite hoje aceder à Internet a velocidades superiores a 8 *Mbit/s* e efectuar chamadas telefónicas em simultâneo.

Tradicionalmente as redes de dados e as redes de voz eram redes completamente separadas, utilizando infraestruturas próprias. Para o surgimento de redes de comunicações integradas capazes de suportar aplicações como voz sobre IP e videoconferência, foi necessário munir essas redes de mecanismos que garantissem a Qualidade de Serviço (QoS) dos fluxos de tráfego. Estas novas aplicações necessitam de garantias de QoS por parte da rede, como por exemplo valores reduzidos de rácio de pacotes perdidos ou do atraso médio dos pacotes, ou de larguras de banda mínimas entre o emissor e o receptor de modo a funcionarem com a qualidade necessária.

A engenharia de tráfego assume uma importância fundamental neste cenário, fornecendo ferramentas essenciais para uma correcta gestão e dimensionamento dos recursos existentes, e estudando os mecanismos essenciais para a implementação da QoS nas redes de comunicações.

O uso de um simulador de redes de comunicações permite consolidar através de simulações a avaliação de desempenho e a optimização das configurações dos diversos mecanismos de rede. O simulador escolhido para efectuar as simulações desta dissertação é o NS (*Network Simulator*). Este simulador é utilizado de forma alargada, tanto pela comunidade científica como pelos operadores, para resolver problemas de desempenho em redes de comunicações.

Entre as vantagens que lhe são apontadas face a outros simuladores, estão o grande número de modelos que já integra, a possibilidade de definir novos modelos e o facto da sua utilização ser livre.

As redes IP operam segundo o paradigma *store-and-forward*, onde os pacotes são armazenados nas filas de espera dos nós e em seguida transmitidos. As filas de espera são um ponto sensível quando se pretende configurar QoS numa rede. Os pacotes podem sofrer atrasos significativos nas filas ou mesmo ser descartados quando as filas ficam cheias. Estas filas de espera possuem algoritmos de escalonamento e de descarte de pacotes, que permitem escolher qual o próximo pacote a ser transmitido e qual o próximo pacote que deve ser descartado (quando necessário), respectivamente. Essas escolhas são essenciais na realização das garantias de QoS pretendidas pelas aplicações.

Na rede, para além da gestão das filas de espera, são necessários mecanismos de controlo de fluxo de modo a garantir a QoS. O controlo de fluxo pode ser efectuado em malha fechada ou em malha aberta. O primeiro é usado para evitar ou resolver problemas de congestionamento, utilizando, por exemplo, o protocolo de transporte TCP (*Transmission Control Protocol*). Os emissores adaptam-se dinamicamente ao estado da rede, recebendo de forma explícita ou implícita sinais de congestionamento da rede. O segundo é usado para garantir que os fluxos respeitem as especificações contratadas, utilizando mecanismos de policiamento. Os pacotes dos fluxos são marcados, consoante o respeito ou não dos contratos, para que nos nós seguintes esses pacotes sofram um tratamento distinto.

O IETF (*Internet Engineering Task Force*) é a entidade responsável pela definição da implementação da QoS nas redes IP. A QoS é definida como sendo um conjunto de características e funcionalidades que dinamicamente controlam e satisfazem requisitos de aplicações e serviços sensíveis a descartes, atrasos e variações dos atrasos (*jitter*) do pacotes. As arquitecturas de QoS definidas pelo IETF co-habitam com o serviço já presente nas redes IP, o *Best-effort*.

1.2 Objectivos da dissertação

Esta dissertação tem como objectivo estudar a problemática da QdS em redes de comunicações com o auxílio do simulador de redes NS-2.

Neste sentido a dissertação estuda os princípios teóricos dos sistemas de filas de espera, das técnicas de descarte, dos algoritmos de escalonamento, dos mecanismos de controlo de congestionamento, dos mecanismos de policiamento e das arquitecturas de QdS, consolidando esses conceitos teóricos através de cenários de simulação.

Um outro objectivo da dissertação é a familiarização com o NS-2 e a sua exploração para fins pedagógicos. Todo o conhecimento adquirido sobre o NS ao longo desta dissertação foi compilado em anexos, que se constituem em pequenos manuais práticos do NS.

1.3 Organização da dissertação

Esta dissertação está organizada em 9 capítulos.

No capítulo 2 é apresentado o simulador NS-2, sendo explorados os conceitos básicos do NS necessários à criação, simulação e análise de uma ligação ponto-a-ponto.

No capítulo 3 são estudados os sistemas de filas de espera M/M/1, M/D/1 e M/G/1 (com e sem prioridades), comparando os valores obtidos das medidas de desempenho destes sistemas em cenários de simulação com os valores teóricos. Neste capítulo também é analisada a validade da aproximação de Kleinrock.

No capítulo 4 são apresentados os algoritmos de escalonamento FIFO (*First-in-first-out*), FQ (*Fair Queuing*), SFQ (*Stochastic Fair Queuing*), DRR (*Deficit Round Robin*) e CBQ (*Class-Based Queuing*). Estes algoritmos são comparados e são analisados os princípios teóricos de funcionamento de cada um.

No capítulo 5 são analisadas as técnicas de descarte. As técnicas de descarte DropTail, DropFront e RED (*Random Early Detection*) são estudadas e comparadas através de simulação, consolidando os conceitos teóricos.

No capítulo 6 é descrita a evolução do protocolo de transporte TCP (*Transmission Control Protocol*) tendo em conta o relógio de retransmissão e os mecanismos de controlo de congestionamento utilizados pelas diversas versões de TCP implementadas.

No capítulo 7 são descritos os mecanismos de policiamento, *Token bucket*, *srTCM* (*Single Rate Three Color Marker*), *trTCM* (*Two Rate Three Color Marker*) e *tswTCM* (*Time Sliding Window Three Colour Marker*). Os mecanismos são analisados através de simulação.

No capítulo 8 são descritas as duas arquiteturas de Qualidade de Serviço (QoS) definidas pelo IETF (*Internet Engineering Task Force*), a *Integrated Services* (IntServ) e a *Differentiated Services* (DiffServ). A arquitetura DiffServ, que surge depois e colmata as limitações identificadas na arquitetura IntServ, é analisada em mais pormenor com a ajuda de simulações.

Finalmente, no capítulo 9 são apresentadas as conclusões do trabalho, assim como linhas orientadoras do trabalho futuro.

2. O simulador, NS-2

O uso de um simulador de redes de comunicações permite consolidar através de simulações o estudo teórico dos mecanismos a aplicar nos nós de uma determinada rede e avaliar qual a melhor configuração de um determinado mecanismo. A simulação tem trazido benefícios às redes de comunicações em geral permitindo (i) estudar e compreender modelos teóricos, (ii) testar e analisar novos mecanismos, (iii) parametrizar e avaliar técnicas presentes, (iv) identificar problemas antes que eles ocorram, (v) reduzir os custos de investigação por não ser necessário investir numa infraestrutura física, (vi) prever resultados na execução de uma determinada acção, (vii) escolher a melhor solução para um determinado cenário e (viii) muitas vezes permite ir onde os modelos teóricos não conseguem.

A simulação é uma ferramenta que permite a partir de um cenário criado orientar um processo de tomada de decisões, analisar e avaliar um sistema proposto, e propor soluções de melhoria de desempenho de um determinado sistema. A constante evolução das redes de comunicações tem sido acompanhada com avanços na área da computação, com computadores mais rápidos e eficientes, novas linguagens de programação, que têm permitido utilizar simuladores nas diversas áreas das redes de comunicações, ajudando ao seu desenvolvimento.

O simulador escolhido, para efectuar as simulações desta dissertação, é o NS (*Network Simulator*). O NS é um simulador de eventos discretos para redes de comutação de pacotes. Este simulador está a ser desenvolvido pelo projecto *Virtual Internet Testbed* (VINT), em que participam a Universidade da Califórnia em Berkeley, o *Lawrence Berkeley National Laboratory* (LBL), o *Information Sciences Institute* (ISI) da Universidade da Califórnia Sul (USC) e o laboratório Xerox PARC. A comunidade científica também contribui para o seu crescimento e melhoramento [1]. Este simulador, distribuído gratuitamente na Internet, é utilizado em inúmeras universidades para o estudo e investigação na área das redes de comunicações.

O NS permite simular uma grande diversidade de redes de comunicações: redes de ligações ponto-a-ponto, redes de área local (*Local Area Networks* - LANs) com e sem fios, redes de comunicações móveis e redes de satélites. Uma grande variedade de protocolos de transporte (TCP, UDP, SRM,...) e de encaminhamento (estático, dinâmico,...) são implementados neste simulador. Nas filas de espera é possível configurar o algoritmo de escalonamento (FIFO, SFQ, DRR,...) e/ou a técnica de descarte (DropTail, RED,...). O simulador disponibiliza para

a transmissão de pacotes fontes de tráfego (CBR, *exponencial ON/OFF*,...) e simula algumas aplicações (FTP, *telnet*,...).

Este capítulo tem por objectivo explorar os conceitos básicos do NS necessários à criação, simulação e análise de uma ligação ponto-a-ponto.

Este capítulo é organizado da seguinte forma. Na secção 2.1 é descrita a estrutura do simulador. Na secção 2.2 todos os elementos de uma topologia de uma rede ponto-a-ponto são analisados, através da construção passo-a-passo um cenário de simulação. A extracção e visualização dos resultados da simulação é referida na secção 2.3. Por fim na secção 2.4 são apresentadas as conclusões deste capítulo.

2.1 Estrutura do NS

O NS é um programa desenvolvido em duas linguagens de programação por objectos, C++ e Otcl. O simulador é programado nessas duas linguagens de modo a separar a parte do controlo da parte de implementação da simulação.

O C++ é utilizado para manipular pacotes e implementar os protocolos, tarefas para as quais a velocidade de execução é vital. Normalmente, a necessidade de efectuar alterações ao código C++ é pontual (a não ser que tenham de ser implementados ou modificados novos elementos ou protocolos). O controlo das simulações necessita de uma linguagem que permite facilmente alterar o código de modo a rapidamente ajustar os parâmetros da simulação, sendo o OTcl utilizado para cumprir esse requisito. O OTcl é utilizado para configurar os parâmetros da simulação e o C++ é utilizado para executar as simulações.

Os passos necessários para a instalação do NS estão descritos no anexo A. Nesse anexo também é feita referência a alguma documentação sobre o simulador. A estrutura de directórios do NS, apresentada na figura 2.1, permite rapidamente encontrar e/ou acrescentar objectos ao simulador.

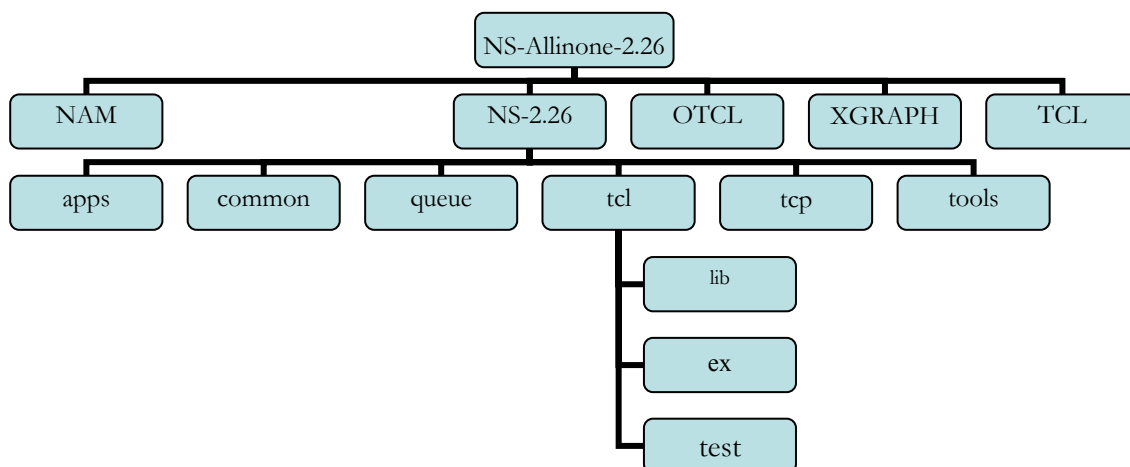


Figura 2.1 – Alguns directórios da estrutura

Todos os objectos C++ estão dentro do directório NS-2.26 divididos em vários sub-directórios. O nome dos sub-directórios dá por vezes a indicação do tipo de objectos que lá estão armazenados. Por exemplo *queue* e *TCP* armazenam objectos relacionados com filas de espera e com o protocolo de transporte TCP. Os objectos relacionados com as redes propriamente ditas, os pacotes e os nós estão agrupados no sub-directório designado por *common*. Outro sub-directório é o *tools* onde, como o próprio nome indica, podemos encontrar ferramentas utilizadas para gerar pacotes e para obter parâmetros de desempenho da rede. Dentro do directório NS-2.26, para além dos sub-directórios com os diversos objectos C++, existe o sub-directório *Tcl* onde se podem encontrar alguns exemplos de *scripts* dentro dos sub-directórios *ex* e *test*. Todas as variáveis e os procedimentos da linguagem de programação OTcl são inicializados dentro sub-directório *lib*.

2.2 Cenário de Simulação

Nesta secção é descrita a construção de um cenário de simulação no NS. O objectivo é construir passo-a-passo um *script* Tcl para a simulação de uma ligação ponto-a-ponto, analisando os seus diversos elementos.

O primeiro objecto criado em qualquer simulação é o *Simulator*, através do qual todos os outros objectos vão ser instanciados. O objecto *Simulator* vai controlar através da sua agenda de eventos discretos toda a simulação.

A simulação de uma ligação ponto-a-ponto é iniciada pela construção da parte física, constituída por nós ligados entre si. Nos dois extremos da ligação, designados por nó origem (emissor) e nó destino (receptor), são colocadas entidades responsáveis por definir o protocolo de transporte (TCP ou UDP) que se designam por agentes. A transmissão de dados é efectuada através de fontes de tráfego e/ou aplicações ligadas aos respectivos agentes (por exemplo FTP sobre TCP ou CBR sobre UDP).

O início e o fim da simulação são definidos na agenda de eventos discretos, bem como os instantes do começo e da finalização da transmissão de dados.

O objectivo da simulação é estimar métricas de desempenho do sistema.

2.2.1 Nó

Um nó representa um emissor, um receptor ou um comutador da rede. Quando um objecto do tipo Nó é criado, são também criados dois outros objectos, um classificador de endereços e um classificador de portos. A função desses classificadores é distribuir correctamente os pacotes que chegam aos agentes e os que partem para as ligações, respectivamente.

O nó criado é identificado numericamente de uma forma sequencial. Cada nó possui uma listagem dos nós vizinhos e outra dos agentes ligados a si. Possui também um módulo de encaminhamento, onde são colocados os classificadores específicos de cada protocolo de encaminhamento.

A figura 2.2 apresenta os primeiros elementos criados: os nós 0 e 1. No anexo B.1 é descrito o comando TCL para criar um nó.



Figura 2.2 – Nós criados

2.2.2 Ligação

A ligação entre dois nós pode ser unidireccional ou bidireccional. Ao criar a ligação é necessário especificar a largura de banda, o algoritmo de escalonamento e/ou técnica de descarte da fila de espera, e o atraso de propagação.

A figura 2.3 apresenta todos os elementos (objectos) que constituem uma ligação unidireccional: *Queue*, *Delay*, *Agent/Null* e *TTL* (*Time To Live*). Estes elementos também existem no sentido oposto no caso da ligação ser bidireccional.

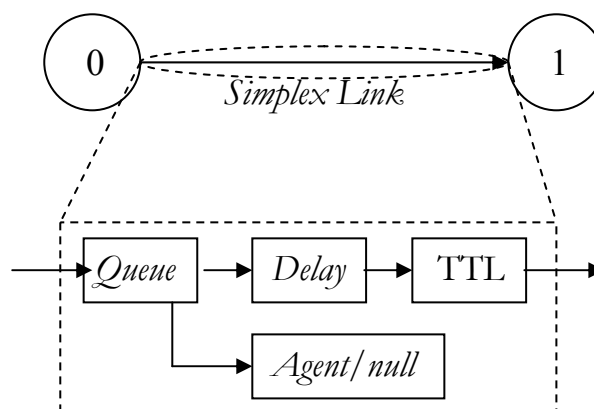


Figura 2.3 – Objectos presentes numa ligação unidireccional

No NS, a fila de espera (*Queue*) não é implementada no nó, mas sim na ligação. Os algoritmos de escalonamento e as técnicas de descarte possíveis de configurar nas filas de espera são descritos nos capítulos 4 e 5, respectivamente.

O objecto *Agent/Null* é utilizado quando existe a necessidade de descartar os pacotes que chegam à fila. O atraso de propagação é simulado através do objecto *Delay*. Por último, o objecto *TTL* actualiza no cabeçalho de cada pacote o valor de *TTL*.

Numa rede existem quatro tipos de atraso: processamento, transmissão, espera na fila de espera e atraso de propagação. No NS apenas o atraso de processamento não é simulado.

Nesta fase, a topologia de rede encontra-se criada e é apresentada na figura 2.4. Na figura, 01 representa a fila de espera da ligação entre o nó 0 e o nó 1 e 10 representa a fila de espera da ligação entre o nó 1 e o nó 0.

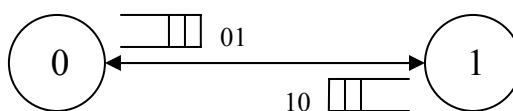


Figura 2.4 – Topologia criada

O comando TCL para criar uma ligação é descrito no anexo B.2.

2.2.3 Agente

Os agentes definem o protocolo de transporte a utilizar, TCP ou UDP, e indicam qual o nó de origem (emissor) e o de destino (receptor) do tráfego. Os agentes são criados aos pares, ou seja, para cada agente de origem é necessário um agente de destino.

O UDP é um protocolo de transporte não orientado à ligação. Neste protocolo não há o conceito de sessão. Os pacotes podem chegar ao destino fora de ordem e não existe garantia de entrega dos pacotes. O TCP é um protocolo de transporte orientado a ligação. Neste protocolo existe o conceito de sessão sendo as ligações estabelecidas através de um processo designado por *three-way-handshaking*. O TCP garante a entrega dos pacotes pela ordem com que foram transmitidos e implementa mecanismos de controlo de fluxos.

No caso do UDP existe apenas um tipo de agente de origem, enquanto que no TCP estão implementadas diversas versões no NS: RFC793, Tahoe, Reno, NewReno, Vegas, Sack1 e FACK. Algumas destas versões são analisadas no capítulo 6.

Os agentes de destino do TCP têm um papel activo no protocolo. Estes, na sua implementação mais simples, geram uma confirmação por cada pacote recebido e enviam-na ao nó de origem de modo a garantir que toda informação transmitida pela origem chegue ao destino e que exista controlo de fluxo dos pacotes. Devido às especificações de cada versão do protocolo de transporte TCP, existem agentes de destino específicos para cada versão de agentes de origem. No caso do UDP os agentes de destino não têm qualquer papel na implementação do protocolo, mas podem ser utilizados para obter informações sobre a rede, como por exemplo número de pacotes recebidos e/ou perdidos.

Os agentes criados no NS estão descritos em pormenor no anexo B.3, permitindo através de uma consulta rápida verificar, os comandos para criar os diversos agentes, todos os seus parâmetros, as variáveis de estado e/ou contadores estatísticos presentes em cada um e possíveis de configurar a partir do *script* Tcl.

No NS os agentes de origem e de destino, depois de criados, são associados aos seus respectivos nós e são ligados entre si. A figura 2.5 apresenta o cenário de simulação onde figuram um agente de origem TCP (scr1), um agente de destino TCP (sink1), um agente de origem UDP (scr0) e um agente de destino UDP (sink0) a partilharem a ligação entre os nós 0 e 1.

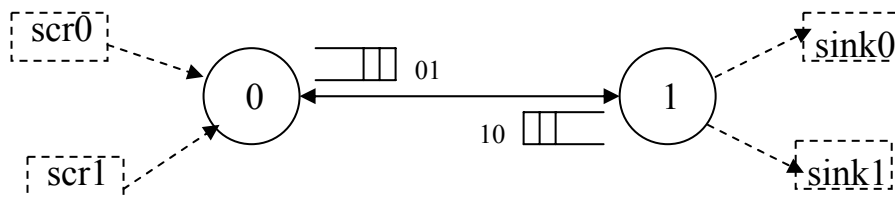


Figura 2.5 – Cenário exemplo para a simulação, introduzidos os agentes

2.2.4 Gerador de Tráfego e Aplicação

O envio de pacotes, entre o nó de origem e o nó de destino no NS, pode ser efectuado através de várias fontes de tráfego e aplicações presentes no simulador. No NS existem quatro tipos de fontes de tráfegos designadas no simulador por *EXPOO_Traffic*, *POO_Traffic*, *CBR_Traffic* e *TrafficTrace*.

A fonte de tráfego designada por *EXPOO_Traffic* gera tráfego *ON/OFF* de acordo com uma distribuição exponencial. Os pacotes são enviados com uma taxa de transmissão fixa durante o período activo (*ON*). No período inactivo (*OFF*) nenhum pacote é enviado. Ambos os tempos a *ON* e a *OFF* são obtidos de acordo com uma distribuição exponencial. O tamanho dos pacotes é fixo. A fonte de tráfego designada por *POO_Traffic*, gera tráfego de acordo com uma distribuição *Pareto*. Os tempos a *ON* e a *OFF* são obtidos de acordo com uma distribuição de *Pareto*. A geração de tráfego com uma taxa de transmissão fixa é obtida através da fonte de tráfego *CBR_Traffic*. Nessa fonte o tamanho dos pacotes é fixo. Por fim, a fonte de tráfego designada por *TrafficTrace*, gera tráfego de acordo com um ficheiro de *trace*. O primeiro campo do ficheiro contém o instante de envio do pacote em micro-segundos. O segundo campo contém o tamanho em *bytes* do pacote.

No decorrer da dissertação surgiu a necessidade de adicionar ao NS uma nova fonte de tráfego, que será descrita em detalhe no capítulo 3, que se designou por *EXP_EXP_Traffic*. Essa nova fonte gera tráfego de pacotes com tamanhos exponencialmente distribuídos, e os intervalos de tempo entre pacotes seguem uma distribuição exponencial.

O NS disponibiliza duas aplicações, uma que simula uma ligação *telnet*, designada no simulador de *Application/Telnet* e uma outra que simula a transferência de uma grande quantidade de dados (FTP), designada no simulador de *Application/FTP*.

O anexo B.4. descreve todos os parâmetros configuráveis nas diversas fontes de tráfego e nas aplicações, assim como os comandos para a sua geração.

Completando o cenário apresentado na figura 2.5, na figura 2.6 incluem-se os elementos *traffic0* e *traffic1* que representam uma fonte de tráfego do tipo *EXPOO_Traffic* e uma aplicação do tipo *Application/Telnet*, respectivamente.

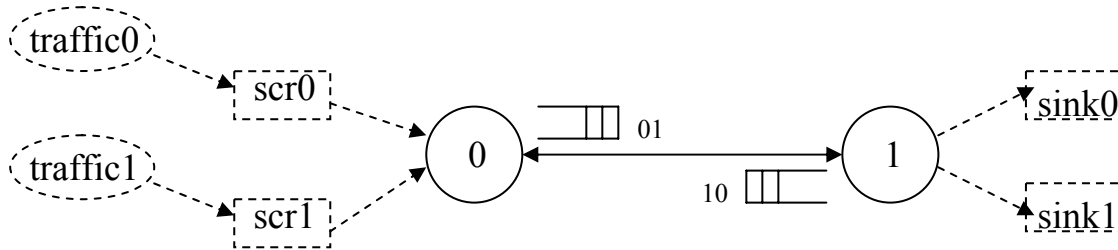


Figura 2.6 – Cenário exemplo para a simulação, introduzidas as fontes de tráfego

2.2.5 Agendar os eventos discretos

A agenda de eventos discretos é criada quando da criação do objecto *Simulator*.

No *script* Tcl define-se o início e o fim da simulação, que correspondem ao primeiro e ao último evento da agenda. Durante a simulação novos eventos podem ser agendados pelos objectos, por exemplo o início e o fim da transmissão de dados.

Esses eventos são colocados e ordenados na agenda de eventos por ordem cronológica. O simulador começa a ler a agenda quando recebe o evento de início de simulação, lê o primeiro evento da agenda, executa o evento e passa ao próximo evento. Repete esses passos até receber o evento que indica o fim da simulação. No anexo B.5 é descrita a sintaxe para agendar eventos no NS.

O tempo no NS não corresponde ao tempo real. Não existe qualquer relação entre a duração da simulação e o instante de tempo agendado para o evento que indica o fim da simulação. A duração da simulação depende do número de eventos que são executados durante o decorrer da simulação.

2.3 Extrair resultados da simulação

O cenário escolhido para a simulação é o da figura 2.6. No NS é possível efectuar análises gráficas e/ou numéricas. A leitura das variáveis de estado dos objectos ou a análise de um ficheiro de registo permitem extrair resultados da simulação.

No NS são disponibilizadas duas ferramentas de visualização: o NAM e o Xgraph. O NAM permite visualizar a topologia da rede e os pacotes da simulação e o Xgraph possibilita a

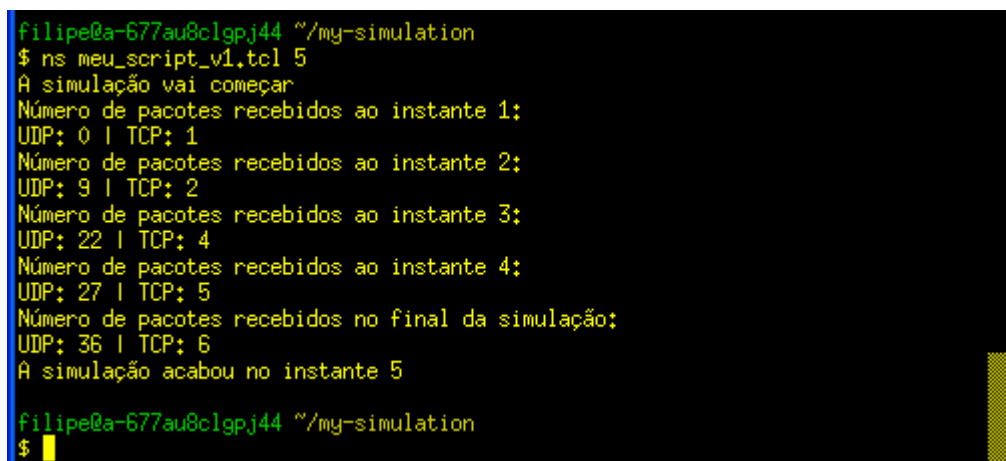
criação de gráficos. O Gnuplot, programa existente no UNIX para criar gráficos, também pode ser utilizado pelo NS.

2.3.1 Leitura das variáveis dos objectos

Uma forma de extrair resultados da simulação é através da leitura das variáveis dos objectos. Estas variáveis são descritas no anexo B.

Se se pretender, por exemplo, saber quantos pacotes cada agente de destino recebeu, no caso do protocolo UDP é necessário ler a variável de estado do agente de destino que indica o número de pacotes recebidos. No caso do protocolo TCP, o agente de destino não possui essa variável de estado mas podemos obter essa informação no agente de origem lendo a variável de estado que indica o número de confirmações (ACK), porque o agente destino envia um ACK por cada pacote recebido.

No anexo D.1 é descrita a sintaxe para ler as variáveis de estado dos objectos. É possível extrair alguns resultados da simulação no final da simulação ou durante a simulação com uma determinada periodicidade. A figura 2.7 apresenta os primeiros resultados da simulação correndo o cenário da figura 2.6.



```
filipe@a-677au8clgpj44 ~/my-simulation
$ ns meu_script_v1.tcl 5
A simulação vai começar
Número de pacotes recebidos ao instante 1:
UDP: 0 | TCP: 1
Número de pacotes recebidos ao instante 2:
UDP: 9 | TCP: 2
Número de pacotes recebidos ao instante 3:
UDP: 22 | TCP: 4
Número de pacotes recebidos ao instante 4:
UDP: 27 | TCP: 5
Número de pacotes recebidos no final da simulação:
UDP: 36 | TCP: 6
A simulação acabou no instante 5

filipe@a-677au8clgpj44 ~/my-simulation
$
```

Figura 2.7 – Primeiros resultados extraídos

Note-se que no anexo B não foi possível descrever todos os objectos presentes no NS. Uma forma rápida de identificar as variáveis e os parâmetros de configuração dos restantes objectos é apresentada no anexo D.2.

2.3.2 Registo de actividade da rede (*Trace*)

No NS existem três tipos de registo de actividade da rede: os registos efectuados nas ligações, os registos de alteração do valor de uma determinada variável e os ficheiros de registos personalizados.

Registos efectuados nas ligações

A passagem dos pacotes pelas ligações pode ser registada e guardada em ficheiro. No anexo D.3 são descritos os comandos para activar essa funcionalidade. Todas as ligações criadas depois de activada essa funcionalidade terão, para além dos objectos já descritos na secção 2.2.2, quatro novos objectos *EnqT*, *DeqT*, *DrpT* e *RecvT*, apresentados na figura 2.8.

O objecto *EnqT* regista o evento da passagem de um pacote que chega à fila de espera, o objecto *DeqT* regista o evento da saída de um pacote da fila de espera, o objecto *RecvT* regista o evento da chegada de um pacote ao fim da ligação e o objecto *DrpT* regista o evento do descarte de um pacote na fila de espera. A passagem de um pacote gera 3 eventos em cada ligação.

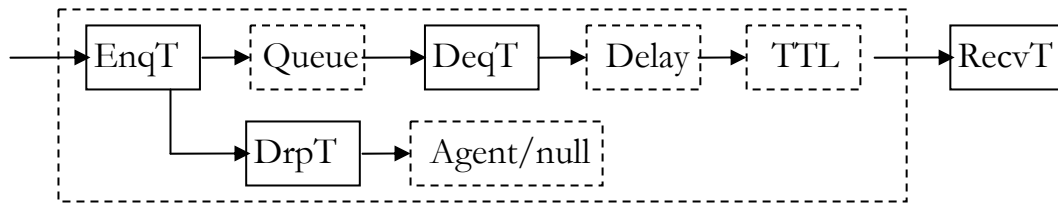


Figura 2.8 – Ligação unidireccional com registo activado

Cada linha do ficheiro criado representa um evento e inclui um conjunto de 12 campos com o formato da figura 2.9.

Event	Time	From node	To node	Pkt type	Pkt size	Flags	Fid	Src addr	Dst addr	Seq num	Pkt id
-------	------	-----------	---------	----------	----------	-------	-----	----------	----------	---------	--------

Figura 2.9 – Formato de cada linha do ficheiro

O campo *Event* indica o tipo de evento. É igual a “+” se o evento for gerado por *EnqT*, igual a “-” para *DeqT*, igual a “d” para *DrpT* e igual a “r” para o evento gerado por *RecvT*. O instante de tempo no qual o evento ocorre é registado no campo *Time*. Os campos *From Node* e *To Node* indicam os nós de origem e de destino da ligação, respectivamente. O tipo de pacote enviado, por exemplo um pacote gerado pelo agente de destino TCP que é do tipo ACK, é identificado no campo *Pkt type*. O tamanho do pacote em *bytes* é registado no campo *Pkt Size*. O NS apenas utiliza uma das *flags*, disponibilizadas no campo *Flags*, a *flag* ECN (*Explicit*

Congestion Notification). O campo *Fid* é utilizado para diferenciar os fluxos. Os campos *Srv addr* e *Dst addr* indicam quais os nós, origem e destino, e quais as portas utilizadas por esses nós para enviar e receber o pacote, respectivamente. O campo *Seq num* indica o número de série do pacote e é sequencial. Todos os pacotes têm um número de sequência associado, até os pacotes do tipo UDP, embora esse campo não seja utilizado na implementação do protocolo. E por último a identificação do pacote, que é única durante a simulação, é colocada no campo *Pkt id*.

Registo de alteração do valor de uma variável

É possível registar num ficheiro todas as alterações de determinadas variáveis de alguns dos objectos do simulador. Para tal essas variáveis devem ser declaradas como *TracedInt* ou *TracedDouble*. No anexo D.3 é apresentado um exemplo do registo das alterações das variáveis de um agente de origem TCP.

A análise desse ficheiro implica uma formatação do mesmo, de modo a permitir extrair a informação relevante para a obtenção de resultados. O tratamento dos ficheiros de registo pode ser efectuado por exemplo pelos programas *awk* e/ou *perl*, que correm em qualquer sistema operativo e são de uso livre. O comando UNIX *grep* também pode ser utilizado para extrair informação do ficheiro, sendo apresentada no anexo D.3 a sintaxe para utilizar esse comando.

Os programas *awk* e *perl* não foram explorados no âmbito deste trabalho. Em simulações mais complexas, estes ficheiros atingem tamanhos gigantescos, ocupando espaço em disco e recursos no processador. Fica aqui só a nota que através de *script awk* ou *perl* é possível extrair informação dos ficheiros de registo de actividade.

Registo personalizados

Existindo a necessidade de efectuar uma análise gráfica da simulação, em vez de utilizar os ficheiros de registo de actividade das ligações ou de alteração de uma variável, é criado um ficheiro e um procedimento que vai registar com uma certa periodicidade os dados pretendidos para a análise. Este ficheiro usado para efectuar a análise gráfica pode ser utilizado directamente como ficheiro de entrada do Xgraph, sem necessidade de recorrer a outros programas (*awk* ou *perl*) para o formatar.

2.3.3 Monitorização da fila de espera

O NS permite monitorizar qualquer fila de espera, de modo a calcular os seus parâmetros de desempenho, como por exemplo calcular o número de pacotes ou *bytes* que chegam e que partem da fila de espera ou saber qual o atraso médio dos pacotes na fila de espera.

O objecto responsável por monitorizar uma fila de espera é o *QueueMonitor*, que por defeito monitoriza a fila com um período de 0.1 s (este valor é configurável). Na criação do objecto *QueueMonitor* são criados mais três objectos apresentados na figura 2.10: *SnoopQ/In*, *SnoopQ/Drop* e *SnoopQ/Out*. Esses objectos notificam o *QueueMonitor* da ocorrência de novos eventos na fila de espera. O objecto *SnoopQ/In* notifica o objecto *QueueMonitor* da chegada de um novo pacote a fila de espera; quando o pacote sai da fila de espera o objecto *QueueMonitor* é notificado pelo objecto *SnoopQ/Out*; sempre que um pacote é descartado da fila de espera o objecto *SnoopQ/Drop* notifica o objecto *QueueMonitor*.

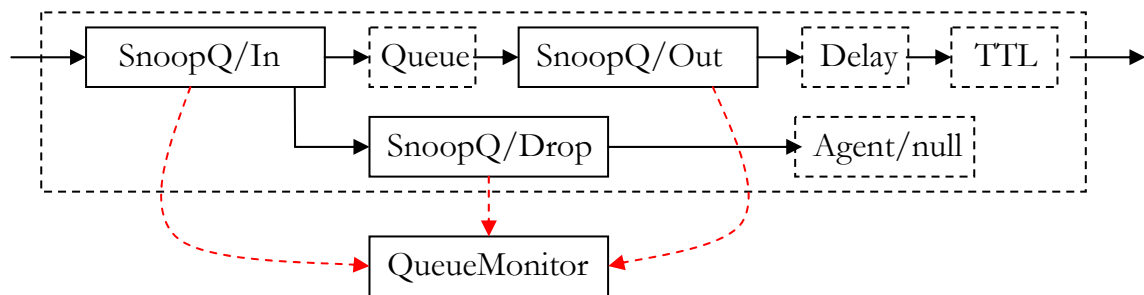


Figura 2.10 – Monitorização de uma fila de espera

A informação recolhida é utilizada pelo objecto *QueueMonitor* para actualizar as diversas variáveis de estado e contadores estatísticos do sistema, por forma a obter os parâmetros de desempenho da fila de espera. Os contadores estatísticos deste objecto permitem obter dados da fila de espera em pacotes e em *bytes*, tais como a quantidade que chegou, a quantidade que saiu e a quantidade de descartados pela fila de espera. Em cada instante é possível saber o número de pacotes e de *bytes* presentes na fila de espera da ligação. Essa informação é colocada nas variáveis do objecto. Este objecto permite também, utilizando os objectos *Integrator* e *Samples*, obter o valor médio das variáveis de estado e o atraso médio na fila de espera.

O objecto *QueueMonitor* monitoriza a fila de espera sem distinguir os fluxos presentes. Os parâmetros de desempenho descritos também podem ser obtidos para cada um dos fluxos que passam pela ligação. Para esse efeito, é colocado à entrada da fila de espera um objecto *classifier* que decide a que fluxo pertence um pacote que chegue. Um objecto *QueueMonitor/ED/Flow* é

criado para cada fluxo que notifica o correspondente objecto *QueueMonitor/ED/Flowmon* da ocorrência de novos eventos. O objecto *QueueMonitor/ED/Flowmon* deriva do objecto *QueueMonitor*, ou seja, todos as variáveis descritas para o *QueueMonitor* também estão disponíveis para cada fluxo.

Este objecto *QueueMonitor/ED/Flow* pode classificar um fluxo de três modos, utilizando os seguintes campos do pacote:

1. *fid*
2. *src_addr + dst_addr*
3. *src_addr + dst_addr + fid*

No primeiro método apenas é utilizado o campo numérico *fid*, presente em cada pacote, para diferenciar os fluxos. O objecto responsável por actualizar esse campo no pacote é o agente, ou seja, é necessário criar um agente distinto por cada fluxo distinto. O segundo método utiliza os campos do pacote que identificam qual o nó de origem e o nó de destino da transmissão: todos os pacotes que são transmitidos de um mesmo nó de origem para um mesmo nó de destino pertencem ao mesmo fluxo. O último método conjuga os dois anteriores, ou seja, um fluxo é identificado pelo seu valor no campo *fid* e pela indicação de qual o nó de origem e o de destino. Ao longo desta dissertação, será utilizado o primeiro método para diferenciar fluxos.

Os parâmetros de desempenho de cada fluxo podem ser obtidos utilizando um ficheiro de *trace* associado ao objecto *QueueMonitor/ED/Flowmon*, que tem o formato apresentado na figura 2.11.

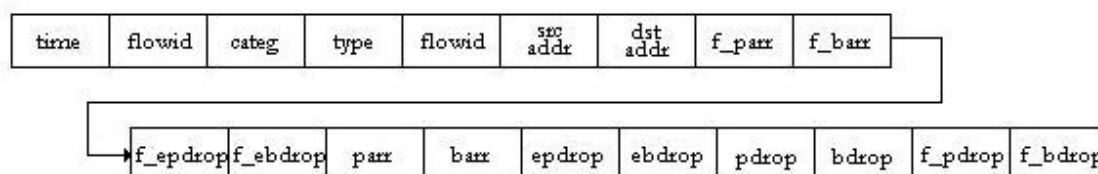


Figura 2.11 – Formato do ficheiro criado pelo objecto *FlowMonitor*

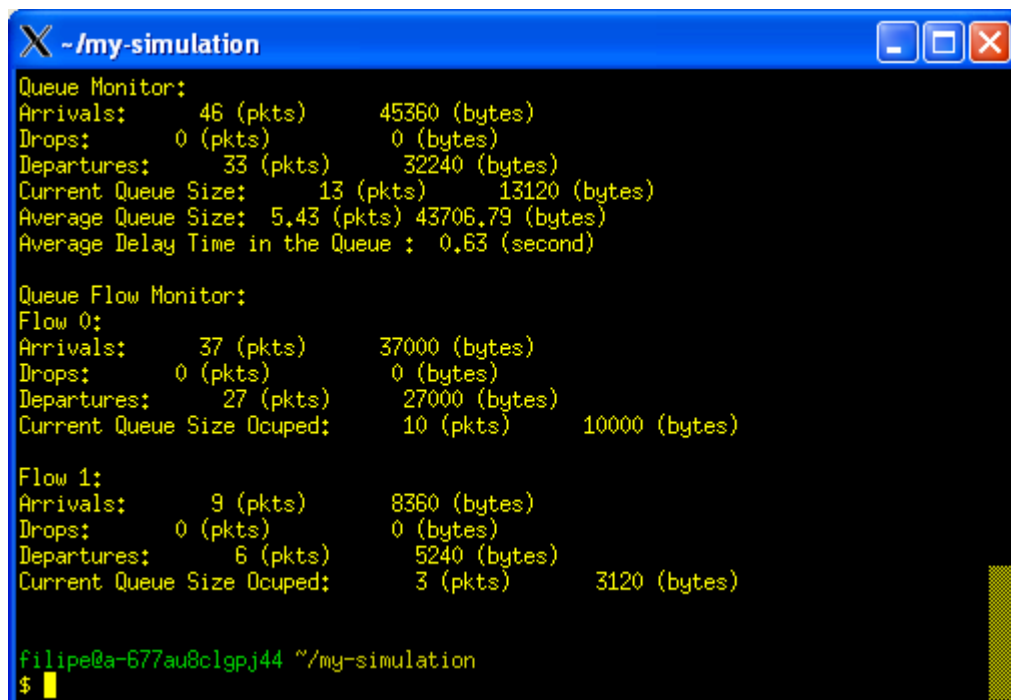
Cada linha do ficheiro possui 19 campos. Os campos iniciados por *f_* são referentes a um determinado fluxo, definido no campo *flowid* e/ou nos campos *src_addr* e *dst_addr*, dependendo do método escolhido para diferenciar os fluxos. Por exemplo os campos *f_parr* e *f_barr* indicam o número de pacotes e de *bytes* deste fluxo que chegaram à fila de espera. Os campos que não são iniciados por *f_* são referentes ao conjunto dos fluxos. Por exemplo os campos

parr e *barr* indicam o número total de pacotes e de *bytes* que chegaram à fila de espera. Todos os campos são descritos no anexo D.4

Formatando o ficheiro é possível efectuar uma análise gráfica e/ou numérica. No entanto este ficheiro não inclui todos os parâmetros de desempenho, em particular as variáveis de estado do tamanho da fila de espera em pacotes e em *bytes*, bem como os contadores estatísticos do número de pacotes e de *bytes* que saíram da fila de espera. Uma solução possível seria alterar o objecto C++ de modo a incluir esses valores no ficheiro. No entanto, esta opção teria como consequência tornar este ficheiro ainda maior, sendo necessário mais processamento para o formatar e obter os resultados pretendidos. Por isso, como já foi mencionado todos os contadores estatísticos e variáveis de estado presentes no *QueueMonitor* também podem ser utilizados no *QueueMonitor/ED/Flowmon* sendo apenas necessário associá-los a cada fluxo e deste modo obter os pretendidos resultados.

O anexo D.4 descreve os objectos utilizados na monitorização da fila de espera, assim como a *sintaxe* dos comandos utilizados.

Activando a monitorização da fila de espera do cenário da figura 2.6, obtém-se o resultado da figura 2.12, diferenciando o fluxo UDP (fluxo 0) do TCP (fluxo 1).



```

~my-simulation
Queue Monitor:
Arrivals: 46 (pkts) 45360 (bytes)
Drops: 0 (pkts) 0 (bytes)
Departures: 33 (pkts) 32240 (bytes)
Current Queue Size: 13 (pkts) 13120 (bytes)
Average Queue Size: 5.43 (pkts) 43706.79 (bytes)
Average Delay Time in the Queue : 0.63 (second)

Queue Flow Monitor:
Flow 0:
Arrivals: 37 (pkts) 37000 (bytes)
Drops: 0 (pkts) 0 (bytes)
Departures: 27 (pkts) 27000 (bytes)
Current Queue Size Ocuped: 10 (pkts) 10000 (bytes)

Flow 1:
Arrivals: 9 (pkts) 8360 (bytes)
Drops: 0 (pkts) 0 (bytes)
Departures: 6 (pkts) 5240 (bytes)
Current Queue Size Ocuped: 3 (pkts) 3120 (bytes)

filipe@a-677au8clgpj44 ~/my-simulation
$

```

Figura 2.12 – Resultado obtidos durante a monitorização da fila de espera

Na figura 2.12 são descritos todos os parâmetros de desempenho possíveis de obter utilizando a monitorização da fila de espera (em pacotes e em *bytes*).

2.3.4 Ferramentas de Visualização

O NS disponibiliza um programa para visualizar a topologia da rede assim como os pacotes transferidos (NAM) e outro para efectuar gráficos (Xgraph). O Gnuplot, programa disponível no UNIX para gerar gráficos, também pode ser utilizado no NS.

NAM

O NAM é baseado num ficheiro de recolha da actividade da rede, criado e activado no *script* Tcl. O anexo D.5 descreve os comandos para utilizar e configurar o NAM. Correndo esse ficheiro obtém-se o resultado apresentado na figura 2.13, que representa uma possível topologia de rede.

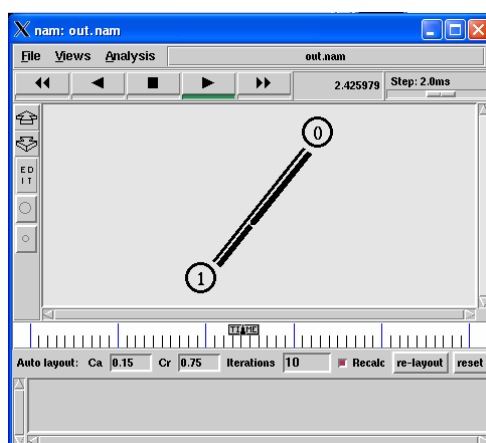


Figura 2.13 - Visualização da topologia da rede

O NAM permite diferenciar por cores todos os elementos da topologia; como por exemplo atribuir uma cor diferente a cada nó e a cada ligação; permite também distinguir por cores os diversos fluxos de pacotes presentes. O NAM possibilita alterar a forma de representação dos nós (circular por defeito), de modo a diferenciar por exemplo os diversos nós emissores dos receptores da topologia. O *layout* da topologia é gerado automaticamente, mas é possível defini-lo manualmente indicando a posição de cada elemento. As filas de espera de cada nó podem ser visualizadas no NAM, assim como os pacotes que são descartados.

A figura 2.14 apresenta uma possível configuração do NAM, onde a forma de um dos símbolos que representa um nó foi alterada e onde é possível visualizar os pacotes de cada fluxo presentes na fila de espera da ligação.

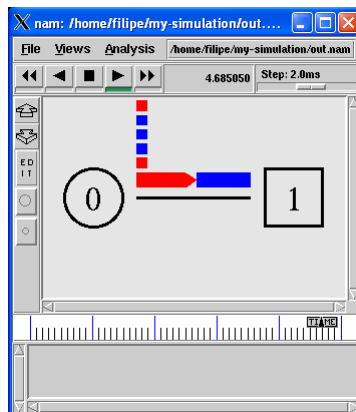


Figura 2.14 - NAM com configurações

Xgraph

O Xgraph é um programa fornecido com o NS para criar gráficos a duas dimensões. É possível gravar os gráficos num dos seguintes formatos: *postscript*, *Tgif*, *HPGL* e *ldraw*. O ficheiro ou os ficheiros de entrada do Xgraph devem conter duas colunas, tendo na primeira os dados da coordenada x e na segunda os da y. A figura 2.15 é um exemplo de um possível gráfico obtido com o Xgraph.



Figura 2.15 – Gráfico do Xgraph

O anexo D.5 descreve o comando e algumas configurações para efectuar gráficos com o Xgraph.

Gnuplot

O Gnuplot é um programa presente no UNIX/LINUX que permite desenhar gráficos a duas ou a três dimensões. O anexo D.4 descreve os passos necessários para utilizar esta ferramenta dentro do *script* Tcl. A figura 2.16 representa o mesmo gráfico que a figura 2.15, mas realizado pelo Gnuplot.

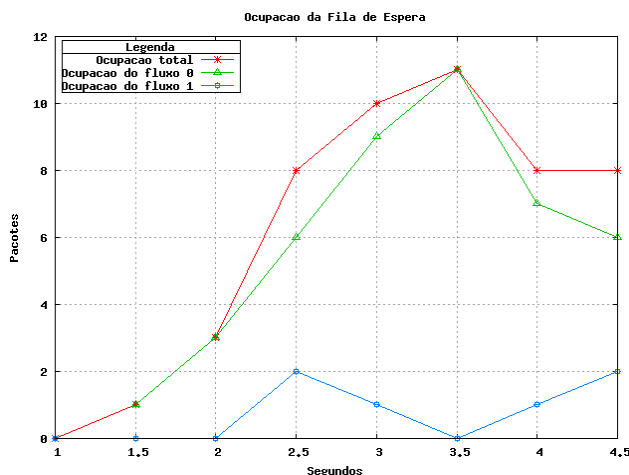


Figura 2.16 – Gráfico do Gnuplot

O Gnuplot é mais versátil que o Xgraph:

- Permite distinguir as curvas não apenas pelas cores, o que é útil quando se imprimem documentos a preto e branco;
- É possível escolher num ficheiro quais as colunas a utilizar para efectuar os gráficos, o que permite ter os dados para vários gráficos num único ficheiro;
- Possui um grande número de possíveis extensões para gravar em ficheiro.

2.4 Conclusões

Este capítulo explora os elementos básicos de uma simulação no NS. Todo o código relacionado com a construção do *script* Tcl, no qual é definido o cenário da simulação, foi colocado em anexo. Os anexos podem ser vistos como um manual de consulta, onde é possível encontrar por exemplo, as variáveis configuráveis de cada objecto e a sintaxe dos comandos para criar os objectos.

Em qualquer *script* Tcl é necessário primeiro, criar um objecto *Simulator* responsável pela gestão de toda simulação. De seguida, formar a topologia de rede escolhendo e configurando os nós e as ligações entre eles. Os agentes são os objectos responsáveis pela definição do protocolo de transporte a utilizar e indicam na topologia a localização dos emissores e dos receptores. A transmissão de dados é efectuada através de fontes de tráfego e/ou aplicações ligadas aos respectivos agentes (por exemplo FTP sobre TCP ou CBR sobre UDP). Por fim é necessário agendar os eventos, início e fim de simulação, bem como os instantes do início e de fim da transmissão de dados.

O objectivo da simulação é extrair resultados, que permitam por exemplo comparar os resultados obtidos em simulação com os teóricos. Para tal é necessário introduzir no *script* Tcl, ferramentas que permitam extrair e visualizar os resultados da simulação. Os objectos C++ possuem variáveis possíveis de configurar e/ou ler a partir do *script* Tcl possibilitando a extracção de resultados. Outra forma de obter resultados é activar o registo de actividade da rede, que gera um ficheiro com dados da rede. O NS disponibiliza uma aplicação para visualizar graficamente os resultados, designada Xgraph. O NAM possibilita visualizar a topologia da rede, bem como a transferência de pacotes entre os elementos da rede.

3. Sistemas de filas de espera

Os sistemas de filas de espera surgem naturalmente quando se estudam redes de comunicações [2]. Quando um pacote chega a um nó da rede, é inicialmente processado de modo a determinar qual a ligação por onde o pacote deve ser enviado. O pacote é então colocado na fila de espera correspondente, à espera de ser transmitido. O tempo que o pacote fica na fila de espera contribui para o atraso total do pacote na rede, que é uma medida de desempenho importante.

O estudo desta e de outras medidas de desempenho pode ser efectuado com auxílio da teoria das filas de espera. A figura 3.1 apresenta um modelo simples de fila de espera, que utiliza apenas um servidor. No contexto das redes de comunicações o servidor corresponde à linha de transmissão caracterizada pela sua capacidade expressa em *bits/s*, os pacotes representam os clientes e o tempo de serviço corresponde ao tempo de transmissão dos pacotes e é igual a L/C , onde L é o tamanho do pacote em *bits* e C é a capacidade de transmissão da ligação em *bits/s*.

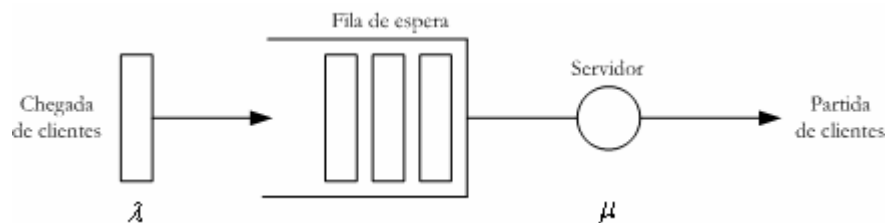


Figura 3.1 – Sistema de uma fila de espera

Na figura o símbolo λ representa a taxa média de chegada de pacotes ($1/\lambda$ representa o tempo médio entre as chegadas dos pacotes) e μ a taxa média de serviço do servidor ($1/\mu$ representa o tempo médio de serviço).

Os sistemas de filas de espera são classificados segundo a notação de Kendall, apresentada na expressão em baixo

$$A/B/c/K \quad (3.1)$$

A especifica a distribuição utilizada para os intervalos entre chegadas, B a distribuição dos tempos de serviço, c o número de servidores e K a capacidade da fila de espera. Os símbolos mais utilizados em A e B são: M – Distribuição *Markoviana* (chegadas de *Poisson* ou intervalos

entre chegadas exponencialmente distribuídos); G – Distribuição genérica (onde o valor da média e da variância são conhecidos); D – Distribuição determinística (constante).

Os sistemas de filas de espera fornecem medidas de desempenho que permitem aos engenheiros de tráfego desenhar sistemas para um determinado serviço tais como, o atraso médio na fila de espera (W_q), o atraso médio no sistema (W), o número médio de pacotes na fila de espera (L_q), e o número médio de pacotes no sistema (L).

O teorema de Little desempenha um papel importante na teoria das filas de espera. Este teorema traduz a ideia intuitiva de que um sistema congestionado (L elevado) está associado a grandes atrasos (W elevado), estabelecendo que

$$L = \lambda W \quad (3.2)$$

Outra expressão utilizada para relacionar as medidas de desempenho é apresentada na equação 3.3, onde conhecido o valor médio da distribuição dos tempos de serviço $E[S]$ e o atraso no sistema é possível calcular o atraso na fila de espera

$$W_q = W - E[S] \quad (3.3)$$

Este capítulo é organizado da seguinte forma. Na secção 3.1 são descritos os princípios teóricos de um sistema M/M/1. A secção 3.2 descreve os princípios teóricos de um sistema M/G/1. A secção 3.3 descreve os princípios teóricos de um sistema M/G/1 com prioridades. A aproximação de Kleinrock é analisada na secção 3.4. A secção 3.5 apresenta os novos objectos inseridos no NS. Os cenários escolhidos para simular os vários sistemas e os resultados obtidos são analisados na secção 3.6. Na secção 3.7 são apresentadas as conclusões deste capítulo.

3.1 M/M/1

O sistema M/M/1 é o mais simples modelo para uma fila de espera, representado por uma fila e um servidor.

Neste modelo os instantes de chegada de clientes são modelados segundo um processo de *Poisson* (M/M/1), os tempos de serviço descritos por uma distribuição exponencial (M/M/1) e existe apenas um servidor com capacidade para servir um cliente de cada vez (M/M/1).

Num processo de *Poisson* com taxa λ , os intervalos entre chegadas são variáveis aleatórias independentes e identicamente distribuídas com distribuição exponencial de média $1/\lambda$.

Um exemplo prático de um sistema M/M/1 é uma ligação ponto-a-ponto com capacidade C , onde chegam pacotes com comprimento L exponencialmente distribuído, a uma taxa de *Poisson* de λ pacotes/s. O tempo de serviço μ é igual a C/L pacotes/s.

As medidas de desempenho deste modelo são dadas por

$$L = \frac{\lambda}{\mu - \lambda} \text{ (número médio de pacotes no sistema)} \quad (3.4)$$

$$W = \frac{L}{\lambda} = \frac{1}{\mu - \lambda} \text{ (atraso médio no sistema)} \quad (3.5)$$

$$W_Q = W - E[S] = W - \frac{1}{\mu} = \frac{\lambda}{\mu(\mu - \lambda)} \text{ (atraso médio na fila de espera)} \quad (3.6)$$

$$L_Q = \lambda W_Q = \frac{\lambda^2}{\mu(\mu - \lambda)} \text{ (número médio de pacotes na fila de espera)} \quad (3.7)$$

3.2 M/G/1

Um dos modelos mais úteis para sistemas de filas de espera é o M/G/1. Relembrando a notação de Kendall, este sistema tem as seguintes características: a chegada de pacotes é um processo de *Poisson* com taxa λ ; o tempo de serviço do servidor segue uma distribuição genérica e independente das chegadas dos pacotes; tem apenas um servidor; e a fila de espera tem tamanho infinito.

Este sistema é mais geral que o M/M/1 estudado na secção anterior, mas esta generalidade tem um preço a pagar. Não é possível extrair as medidas de desempenho deste sistema sem conhecer o valor médio $E[S]$ e o segundo momento $E[S^2]$ da distribuição genérica do tempo de serviço. Sendo conhecidos esses dois valores, a fórmula de Pollaczek-Khintchine indica que o atraso médio na fila de espera é dado por

$$W_Q = \frac{\lambda E[S^2]}{2(1 - \lambda E[S])} \quad (3.8)$$

Quando o tempo de serviço é exponencialmente distribuído, o sistema resulta num M/M/1. Neste caso, o valor médio e o segundo momento da distribuição genérica são dados por

$$E[S] = \frac{1}{\mu} \quad (3.9)$$

$$E[S^2] = \text{var}[S] + E[S]^2 = \frac{1}{\mu^2} + \frac{1}{\mu^2} = \frac{2}{\mu^2} \quad (3.10)$$

Substituindo na fórmula Pollaczek-Khintchine obtém-se, como seria de esperar, a expressão 3.8 relativa ao atraso médio na fila de espera de um sistema M/M/1, ou seja,

$$W_Q = \frac{\lambda}{\mu(\mu - \lambda)} \quad (3.11)$$

Quando o tempo de serviço é igual para todos os pacotes com valor $1/\mu$, o sistema M/G/1 reduz-se a um M/D/1. Neste caso, o valor médio e o segundo momento da distribuição genérica são dados por

$$E[S] = \frac{1}{\mu} \quad (3.12)$$

$$E[S^2] = \text{var}[S] + E[S]^2 = 0 + \frac{1}{\mu^2} = \frac{1}{\mu^2} \quad (3.13)$$

Substituindo na fórmula Pollaczek-Khintchine obtém-se o atraso médio na fila de espera

$$W_Q = \frac{\lambda}{2\mu(\mu - \lambda)} \quad (3.14)$$

Analisando estes dois casos particulares de um sistema M/G/1 verifica-se que, alterando o tamanho dos pacotes de fixo para exponencialmente distribuído, o valor do atraso na fila de espera para um sistema M/D/1 é metade que o valor de um sistema M/M/1.

3.3 Sistema M/G/1 com prioridades

Em muitos sistemas de filas de espera os pacotes são divididos por classes. Uma classe de serviço identifica um conjunto de fluxos de pacotes com as mesmas características de tráfego e os mesmos requisitos de qualidade de serviço. Atribuindo prioridades aos fluxos de uma classe, os pacotes dos fluxos de maior prioridade são sempre servidos antes do que os pacotes de um fluxo de prioridade inferior. Os pacotes dos fluxos com a mesma prioridade são servidos por ordem de chegada (FIFO).

Quando um pacote de maior prioridade chega e um de menor prioridade está a ser transmitido, duas situações podem ocorrer: a transmissão do pacote não é interrompida pela chegada de um pacote de maior prioridade, disciplina de serviço designada por não-preemptiva; ou a transmissão do pacote é interrompida pela chegada de um pacote de maior prioridade, disciplina de serviço designada por preemptiva.

O atraso médio na fila de espera de um sistema M/G/1 do tipo não-preemptivo correspondente à classe K é dado por

$$W_{Q_K} = \frac{\sum_{i=1}^n \lambda_i E[S_i^2]}{2(1 - \rho_1 - \dots - \rho_{K-1})(1 - \rho_1 - \dots - \rho_K)} \quad (3.15)$$

onde $\rho_1 + \dots + \rho_n < 1$ e $\rho_K = \frac{\lambda_K}{\mu_K}$.

Se os tempos de serviço de cada classe forem exponencialmente distribuídos com média $1/\mu$ então esse sistema resulta num M/M/1 do tipo não-preemptivo e o atraso médio na fila da classe K é dado por

$$W_{Q_K} = \frac{\rho_K / \mu}{(1 - \rho_1 - \dots - \rho_{K-1})(1 - \rho_1 - \dots - \rho_K)} \quad (3.16)$$

Se os tempos de serviço forem determinísticos e iguais a $1/\mu$ então esse sistema resulta num M/D/1 do tipo não-preemptivo e o atraso médio na fila da classe K é dado por

$$W_{Q_k} = \frac{\rho/2\mu}{(1-\rho_1-\dots-\rho_{K-1})(1-\rho_1-\dots-\rho_K)} \quad (3.17)$$

3.4 Aproximação de Kleinrock

Nas redes de dados existem filas de espera, as quais interagem umas com as outras. Este efeito torna mais complicado modelar analiticamente os sistemas, devido a correlação existente entre os instantes de chegada e os tempos de serviço dos pacotes. Como exemplo, podemos considerar a linha de transmissão da figura 3.3, onde as ligações têm a mesma capacidade.

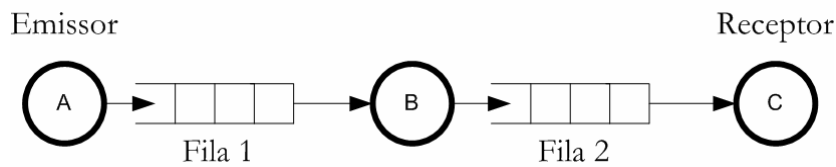


Figura 3.3 – Filas de espera consecutivas

Assumindo que o tamanho dos pacotes é fixo e que o processo de chegadas é de *Poisson*, a primeira fila de espera pode ser modelada por um sistema M/D/1. Uma vez que o tempo de serviço é o mesmo nas duas ligações não existem pacotes na segunda fila de espera. Assim sendo, a segunda fila de espera não é um sistema M/D/1. Considerando um segundo exemplo, em que o tamanho dos pacotes é exponencialmente distribuído e o processo de chegada é de *Poisson*, a primeira fila de espera pode ser modelada como um sistema M/M/1. No entanto, o intervalo entre a chegada de dois pacotes consecutivos na segunda fila é maior ou igual ao intervalo de tempo de serviço do primeiro pacote. Logo a 2ª fila não pode ser modelada por um sistema M/M/1.

A aproximação de Kleinrock consiste em admitir que cada ligação pode ser modelada por um sistema M/M/1, permitindo obter um valor aproximado do número médio de pacotes no sistema e do atraso médio sofrido por estes pacotes. Considere-se a rede da figura 3.4 onde existem vários fluxos de pacotes cada um seguindo por um único percurso. Seja x_s a taxa de chegada do fluxo s ao sistema. A taxa total de chegada à ligação (A, B) é a soma da taxa de chegada de todos os fluxos que passam por essa ligação. Para uma rede de datagramas é necessário utilizar um modelo mais geral, pois nesse tipo de rede os pacotes de um mesmo fluxo podem seguir caminhos distintos na rede, mas a ideia básica é a mesma do modelo de

circuitos virtuais. Como exemplo considere a ligação (A, B) da figura 3.4. A taxa de chegada de pacotes nessa ligação é igual $\lambda_{(A,B)} = x_1 + x_3$.

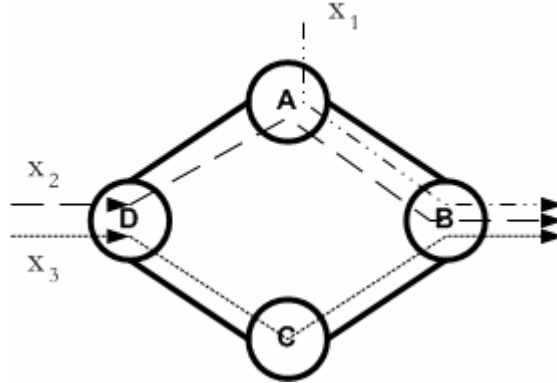


Figura 3.4 – Exemplo aproximação de Kleinrock

A aproximação de Kleinrock sugere que ao juntar vários fluxos de pacotes numa única ligação, a independência entre os tempos de chegada e o tamanho dos pacotes é restaurada. Os intervalos entre chegadas de pacotes tendem a ser exponencialmente distribuídos. Este aproximação conclui que o modelo adequado para cada ligação é o sistema M/M/1. Generalizando a equação 3.4, pode-se aproximar o número médio de pacotes em cada ligação (i,j) por

$$L_{ij} = \frac{\lambda_{ij}}{\mu_{ij} - \lambda_{ij}} \quad (3.18)$$

O número médio de pacotes no sistema obtém-se somando os valores correspondentes de todas as ligações, ou seja,

$$L = \sum_{(i,j)} L_{ij} = \sum_{(i,j)} \frac{\lambda_{ij}}{\mu_{ij} - \lambda_{ij}} \quad (3.19)$$

Utilizando o teorema de Little, obtém-se o valor aproximado do atraso médio dos pacotes no sistema,

$$W = \frac{1}{\gamma} \sum_{(i,j)} \frac{\lambda_{ij}}{\mu_{ij} - \lambda_{ij}} \quad (3.20)$$

onde γ é a soma da taxa de chegada de todos os fluxos que chegam ao sistema. Nos casos em que os atrasos de processamento nos nós de comutação e os atrasos de propagação nas ligações não são desprezáveis, o atraso médio por pacote é dado por

$$W = \frac{1}{\gamma} \sum_{(i,j)} \left(\frac{\lambda_{ij}}{\mu_{ij} - \lambda_{ij}} + \lambda_{ij} d_{ij} \right) \quad (3.21)$$

em que d_{ij} é o atraso médio de processamento e propagação na ligação (i, j) .

No caso em que a cada fluxo está associado um percurso único na rede (circuito virtual), o atraso médio por pacote do fluxo s é dado por

$$W_s = \sum_{(i,j) \in R_s} \left(\frac{1}{\mu_{ij} - \lambda_{ij}} + d_{ij} \right) \quad (3.22)$$

onde o conjunto R_s representa todas as ligações (i, j) de um único percurso do fluxo s na rede.

A maior fonte de erro associada à aproximação de Kleinrock deve-se à correlação entre os comprimentos dos pacotes e os intervalos entre chegadas.

3.5 Objectos adicionados ao NS

No decorrer das simulações deste capítulo foi necessário introduzir no NS dois novos objectos, um novo agente e uma nova fonte de tráfego, de modo a poder atingir os objectivos deste capítulo. A sub-secção 3.5.1 apresenta a nova fonte de tráfego criada, que permite o envio de pacotes com intervalos exponencialmente distribuídos e tamanho exponencialmente distribuído. O novo agente criado, que permite extrair todos os parâmetros de desempenho do sistema, é descrito na sub-secção 3.5.2.

3.5.1 Nova fonte de tráfego Exp_Exp

Um método possível de gerar pacotes para simular um sistema M/M/1 é criar dentro de um *script* Tcl um procedimento com duas variáveis aleatórias, *expo1* e *expo2*, independentes e identicamente distribuídas com distribuição exponencial de média $1/\lambda$ e média b (tamanho do pacote), respectivamente. Um pacote é enviado de cada vez que o procedimento é

chamado, onde a variável *expo1* define a periodicidade com que o procedimento é chamado e *expo2* o tamanho do pacote enviado. O código do procedimento está no anexo F.1.

Outro método para enviar pacotes é utilizar as fontes de tráfego presentes no NS descritas na secção 2.2.4. Segundo o manual do NS [1] um processo de *Poisson* pode ser simulado com uma fonte de tráfego exponencial. A configuração dessa fonte é descrita no anexo B.4. Como o tamanho dos pacotes é fixo, o tempo de serviço do sistema simulado não é exponencialmente distribuído mas sim constante, logo com essa fonte simula um sistema M/D/1 e não o pretendido M/M/1.

Surge então a necessidade de criar uma fonte de tráfego que para além de gerar pacotes com intervalos que seguem uma distribuição exponencial, gere também pacotes com tamanhos exponencialmente distribuídos. As fontes de tráfego presentes no NS já foram descritas no capítulo 2: *EXPOO_Traffic* [Application/Traffic/Exponential]; *POO_Traffic* [Application/Traffic/Pareto]; *CBR_Traffic* [Application/Traffic/CBR]; *TrafficTrace* [Application/Traffic/Trace]. Com a excepção da quarta, todas as fontes geram pacotes de tamanho fixo. A quarta gera tráfego de acordo com um ficheiro de *trace*, em que o tamanho de cada pacote é definido no segundo parâmetro de cada linha do ficheiro. O primeiro parâmetro de cada linha do ficheiro indica o instante de transmissão do pacote.

A nova fonte de tráfego foi especificada para gerar tráfego, de pacotes com tamanhos exponencialmente distribuídos com média igual ao valor indicado na variável *packetSize_* (bytes) e com intervalos de tempo entre pacotes seguindo uma distribuição exponencial com média igual ao valor da variável *intervalTime_* (segundos). A nova fonte é designada por *EXP_EXP_Traffic* [Application/Traffic/Exp_Exp]. Os parâmetros da nova fonte de tráfego são descritos no anexo B.4. O código da nova fonte de tráfego é apresentado no Anexo G.1.

Variáveis aleatórias

A nova fonte de tráfego possui um comando designado por *use-rng*, que permite definir qual a *seed* inicial do gerador de variáveis aleatórias (Ver anexo F.1). Na repetição de simulações se o valor inicial do *seed* não for alterado, as variáveis aleatórias serão geradas sempre com a mesma sequência, o que não é o pretendido.

O anexo G.3 descreve os passos necessários para integrar este objecto no NS.

3.5.2 Novo agente para estimar medidas de desempenho do sistema

Os parâmetros de desempenho da fila de espera são obtidos utilizando o objecto *QueueMonitor* e o objecto *Integrator*, ambos descritos no anexo D.3. Apenas três das quatro medidas de desempenho são possíveis de extrair com esses dois objectos, faltando o atraso médio no sistema. O código Tcl para o cálculo dos parâmetros de desempenho é descrito no anexo F.2. Para obter a quarta medida foi necessário adicionar um novo agente ao NS.

O novo agente surge como um melhoramento de um já existente, acrescentando-lhe novas funcionalidades e alterando outra já existente. O NS possui dois tipos de agentes destino para UDP, o agente *Null* e o agente *LossMonitor*, descritos no Capítulo 2. O agente *Null* tem apenas a função de descartar os pacotes que recebe, pelo que não permite obter parâmetros de desempenho da rede. Por outro lado o agente *LossMonitor* contém contadores estatísticos que permitem calcular a quantidade de pacotes e de *bytes* que chegam do nó origem. Esse agente sabe, através do número de sequência de cada pacote, quantos pacotes foram descartados. Utilizando novamente o número de sequência calcula o tempo em segundos em que chegou o último pacote e qual o próximo pacote esperado. O novo objecto, designado por *NodeMonitor* possui todas as funcionalidades presentes no *LossMonitor*, altera o algoritmo de cálculo de pacotes perdidos e acrescenta um novo procedimento para a recolha do instante de envio do pacote, que permite calcular o atraso máximo, o médio e o mínimo no sistema.

Cálculo do número de pacotes perdidos

Analisando o objecto C++, *lossmonitor.cc* verifica-se como os pacotes perdidos são calculados.

```
/*
 * Check for lost packets
 */
if (expected_ >= 0) {
    int loss = seqno_ - expected_;
    if (loss > 0) {
        nlost_ += loss;
        Tcl::instance().evalf("%s log-loss", name());
    }
}
```

Este objecto calcula os pacotes perdidos subtraindo ao número de sequência do pacote que chega o valor do número de sequência esperado, por exemplo se o próximo pacote esperado (que é igual ao número de sequência do último pacote recebido + 1) for o número 20 e o número de sequência do pacote que chega é 25, isso quer dizer que se perderam 5 pacotes. Desta forma o objecto actualiza o seu contador estatístico de número de pacotes perdidos. Identifica-se uma lacuna na actualização deste contador. O que acontece se um dos

pressupostos pacotes perdidos esteja atrasado e finalmente chega ao nó de destino? O contador obtém um valor negativo, visto que o número de sequência do pacote que chega atrasado é inferior ao valor do esperado. Este algoritmo foi alterado no novo objecto de modo a superar essa lacuna. Quando um pacote chega atrasado, o contador estatístico de pacotes perdidos é simplesmente decrementado.

```
/*
 * Check for lost packets
 */
int loss = seqno_ - expected_;
if (loss == 0) {
    expected_ = seqno_ + 1;
}
else {
    if (loss > 0){
        nlost_ += loss;
        expected_ = seqno_ + 1;
    }
    else {
        nlost_ = nlost_ - 1;
        printf ("Chegou um pacote atrasado");
    }
}
```

Recolha do instante da transmissão do pacote

A ideia é recolher o instante de tempo da transmissão do pacote, de modo a poder calcular parâmetros de desempenho da rede, tais como, o atraso mínimo, o atraso máximo e o atraso médio.

O cabeçalho comum dos pacotes foi analisado à procura de um campo que pudesse ser utilizado por este novo objecto. Utilizar um parâmetro do cabeçalho comum permite que este novo objecto seja flexível e possa ser utilizado independentemente do tipo de pacote transmitido. A estrutura dos pacotes está definida no ficheiro *common/packet.h*. Cada pacote é constituído por um conjunto de cabeçalhos e um campo opcional para dados (ver figura 3.2). Sempre que é criado um pacote, independentemente da aplicação, o pacote é criado com todos os cabeçalhos e é registado o valor do *offset* para cada cabeçalho. O valor de *offset* representa a distância de um cabeçalho ao cabeçalho comum. Qualquer objecto da rede pode aceder aos diversos cabeçalhos do pacote através do valor do *offset* correspondente.

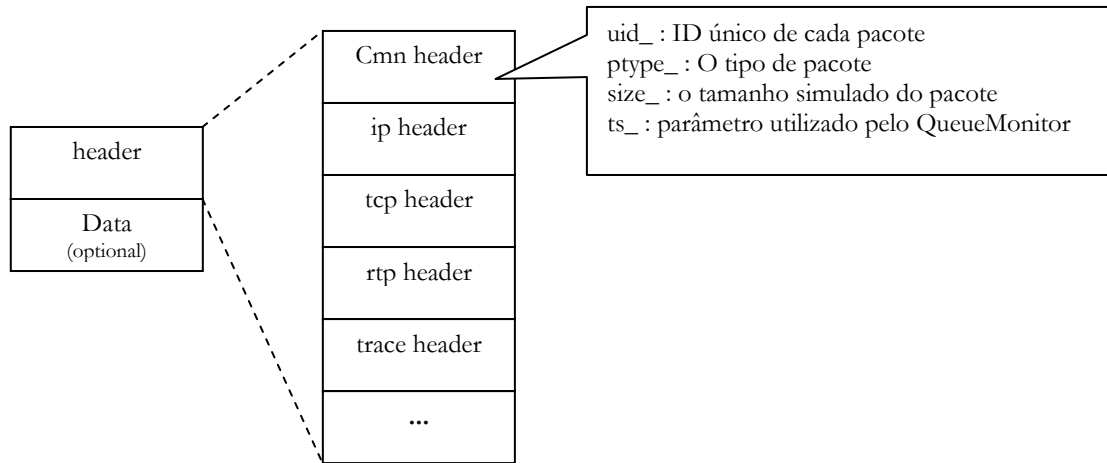


Figura 3.2 – Formato do pacote no NS

Normalmente os pacotes criados apenas possuem os cabeçalhos, porque poucos agentes ou aplicações do NS utilizam o campo de dados, mas este está disponível e pode ser utilizado por novos agentes e/ou aplicações. Os cabeçalhos apresentados na figura 3.2 apenas representam alguns dos cabeçalhos utilizados, sendo sempre que se justifique possível criar um novo cabeçalho para uma determinada aplicação e/ou agente.

No cabeçalho comum foi identificada a variável *txtime_* e a função *txtime()*, que podem ser utilizadas pelo novo objecto. Escolhido o campo a utilizar, este valor tem que ser actualizado no instante da transmissão. O agente é o elemento da rede responsável por actualizar este campo, sendo o UDP o agente de origem do agente *NodeMonitor* (descrito no capítulo 2). No anexo G.2 são descritas as alterações a efectuar necessárias.

O novo objecto só tem que ler o valor do instante de partida e o de chegada do pacote para calcular os parâmetros de desempenho. A função *recv()* do novo objecto é a função responsável pela leitura desses valores e pela actualização das variáveis de estado (ver anexo G.2). Um objecto auxiliar, o *Samples*, é utilizado para o cálculo do atraso médio. Este objecto já foi descrito no capítulo 2 na alínea 2.3.3. O anexo B.3. descreve como activar essa funcionalidade.

O anexo G.3 descreve os passos necessários para integrar este objecto no NS.

3.6 Simulações

Esta secção apresenta os resultados obtidos na simulação dos sistemas M/M/1, M/D/1, M/G/1 sem e com prioridades e no estudo da validade da aproximação de Kleinrock, calculados com intervalos de confiança de 90%. Cada simulação foi repetida 10 vezes e o tempo de duração de cada uma é de 1000 segundos.

Na sub-secção 3.6.1 são apresentados os resultados obtidos na simulação de um sistema M/M/1. A sub-secção 3.6.2 apresenta os resultados obtidos na simulação de um sistema M/G/1. Os resultados relativos à simulação de um sistema M/G/1 com prioridades são apresentados na sub-secção 3.6.3. Por fim a sub-secção 3.6.4 apresenta os resultados da simulação para o estudo da validade da aproximação de Kleinrock.

3.6.1 Sistema M/M/1

A rede escolhida para simular um sistema M/M/1 é muito simples, consiste em dois nós ligados com uma ligação unidireccional, onde existe uma fila de espera do tipo FIFO com tamanho limite de 100000 pacotes e um fluxo de dados entre os dois nós com uma taxa de λ pacotes/s. A largura de banda da ligação (C) em *bit/s* é dada por $C = \mu \times L \times 8$, onde μ é o valor da taxa de serviço escolhida e L é o tamanho dos pacotes em *bytes* (500 *bytes*). O protocolo de transporte escolhido é o UDP.

Pretende-se extrair da simulação os quatro parâmetros de desempenho enunciados para um sistema M/M/1. Os parâmetros de entrada da simulação são a taxa de envio λ e a taxa de serviço μ . O cenário é simulado para vários valores de $\rho = \lambda/\mu$.

A tabela 3.1 apresenta os resultados obtidos para o sistema M/M/1.

λ	μ	Desemp.	Valor Teórico	Valor Simulado
1	2	L	1	[0,995; 1,065]
		W	1	[0,999; 1,070]
		L _Q	0,5	[0,496; 0,557]
		W _Q	0,5	[0,495; 0,554]
0.5	1	L	1	[0,956; 1,067]
		W	2	[1,918; 2,122]
		L _Q	0,5	[0,452; 0,545]
		W _Q	1	[0,903; 1,083]
5	8	L	1,667	[1,541; 1,688]
		W	0,333	[0,315; 0,339]
		L _Q	1,042	[0,952; 1,069]
		W _Q	0,208	[0,191; 0,213]
9	10	L	9	[7,456; 9,077]
		W	1	[0,838; 1,017]
		L _Q	8,1	[6,601; 8,209]
		W _Q	0,9	[0,738; 0,917]

Tabela 3.1 – Simulação de um sistema M/M/1

Todos os valores teóricos estão dentro dos respectivos intervalos de confiança. Esta simulação permite comprovar o correcto funcionamento dos dois novos objectos criados, a nova fonte de tráfego e o novo agente.

3.6.2 Sistema M/G/1

Para o estudo por simulação de um sistema M/G/1 foram escolhidos dois cenários: o primeiro é um caso particular (sistema M/D/1) e o segundo um sistema M/G/1 genérico.

O objectivo desta simulação é constatar o enunciado na fórmula Pollaczek-Khintchine, verificando que neste sistema o atraso na fila de espera é metade do valor que o atraso no sistema M/M/1. Na sub-secção 3.5.1 foi referido como simular um sistema M/D/1. A rede consiste em dois nós ligados com uma ligação unidireccional, onde existe uma fila de espera do tipo FIFO com tamanho limite de 100000 pacotes e um fluxo de dados entre os dois nós com uma taxa de λ pacotes/s. O valor da largura de banda da ligação (C) em *bit/s* é optida através da expressão 3.10, onde μ é o valor da taxa de serviço escolhida e L é o tamanho dos pacotes em *bytes* (500 *bytes*). O protocolo de transporte escolhido é o UDP. O cenário escolhido para simular um sistema M/D/1 é o mesmo cenário que é utilizado para simular o sistema M/M/1, alterando apenas a fonte de tráfego. Os parâmetros de entrada da simulação são a taxa de envio λ e a taxa de serviço μ . O cenário é simulado para vários valores de $\rho = \lambda/\mu$.

A tabela 3.2 apresenta os resultados obtidos para o sistema M/D/1.

λ	μ	Desemp.	Valor Teórico	Valor Simulado
1	2	L	0,750	[0,718; 0,773]
		W	0,750	[0,726; 0,771]
		L _Q	0,250	[0,226; 0,274]
		W _Q	0,250	[0,226; 0,271]
0.5	1	L	0,750	[0,710; 0,760]
		W	1,500	[1,446; 1,514]
		L _Q	0,250	[0,221; 0,256]
		W _Q	0,500	[0,446; 0,513]
5	8	L	1,146	[1,123; 1,168]
		W	0,229	[0,226; 0,233]
		L _Q	0,521	[0,504; 0,545]
		W _Q	0,104	[0,101; 0,108]
9	10	L	4,950	[4,554; 5,357]
		W	0,550	[0,508; 0,594]
		L _Q	4,050	[3,665; 4,467]
		W _Q	0,450	[0,408; 0,494]

Tabela 3.2 – Simulação de um sistema M/D/1

Todos os valores teóricos estão dentro dos respectivos intervalos de confiança. Comparando a tabela 3.2 com a tabela 3.1, verifica-se que o valor do atraso médio na fila de espera de um sistema M/D/1 é metade do valor do atraso de um sistema M/M/1, tal como anunciado na fórmula Pollaczek-Khintchine.

O segundo cenário escolhido simula um sistema M/G/1 genérico. O cenário escolhido consiste em dois nós ligados com uma ligação unidireccional com capacidade igual a 64 Kbits/s (C). A fila de espera presente é do tipo FIFO e com tamanho limite de 100000 pacotes. O protocolo de transporte escolhido é o UDP. A informação entre os dois nós é transmitida via 2 fluxos de dados, caracterizados pela taxa de transmissão λ_i e pelo tamanho fixo dos seus pacotes L_i , onde i indica qual o fluxo. O tempo médio de serviço de cada fluxo (S_i) e a probabilidade de um pacote (P_i) ser de um dos fluxos são calculados através das seguintes expressões

$$S_i = \frac{L_i}{C} \quad (3.23)$$

$$P_i = \frac{\lambda_i}{\sum_{k=1}^2 \lambda_k} \quad (3.24)$$

De acordo com a fórmula Pollaczek-Khintchine é necessário conhecer o valor médio $E[S]$ e o segundo momento $E[S^2]$ da distribuição genérica, calculados através das seguintes expressões

$$E[S] = P_1 S_1 + P_2 S_2 \quad (3.25)$$

$$E[S^2] = P_1 S_1^2 + P_2 S_2^2 \quad (3.36)$$

Os parâmetros de entrada da simulação são a taxa de envio λ e o tamanho dos pacotes L de cada fluxo. O cenário é simulado para vários valores de λ e de L .

Os resultados obtidos para o sistema M/G/1 são apresentados na tabela 3.3.

λ_1	L_1	λ_2	L_2	Desemp.	Valor Teórico	Valor Simulado
1	500	2	1000	L	0,389	[0,370; 0,392]
				W	0,130	[0,128; 0,131]
				L_Q	0,077	[0,072; 0,081]
				W_Q	0,026	[0,024; 0,027]
2	500	1	1000	L	0,297	[0,284; 0,308]
				W	0,099	[0,098; 0,099]
				L_Q	0,047	[0,045; 0,048]
				W_Q	0,016	[0,015; 0,016]
4	500	5	1000	L	4,250	[4,020; 4,417]
				W	0,472	[0,449; 0,489]
				L_Q	3,375	[3,156; 3,548]
				W_Q	0,375	[0,352; 0,392]

Tabela 3.3 – Simulação de um sistema M/G/1

Todos os valores teóricos estão dentro dos respectivos intervalos de confiança.

3.6.3 Sistema M/G/1 com prioridades

O cenário escolhido consiste em dois nós ligados com uma ligação unidireccional com capacidade igual a 64 Kbits/s (C). A fila de espera presente é do tipo PQ e com tamanho limite de 100000 pacotes. O protocolo de transporte escolhido é o UDP. A informação entre os dois nós é transmitida via 2 fluxos de dados, caracterizados pela taxa de transmissão λ_i e pelo tamanho fixo dos seus pacotes L_i , onde i indica qual o fluxo. A cada fluxo é atribuído uma prioridade, sendo o fluxo 1 mais prioritário que o fluxo 2. O tempo médio de serviço de cada fluxo (S) é dado pela expressão 3.27. A probabilidade de um pacote (P_i) ser de um dos fluxos é calculada através da expressão 3.28. Utilizando a expressão 3.18 é calculado o atraso médio na fila de espera de cada fluxo, onde o segundo momento $E[S^2]$ da distribuição genérica é dado

pela expressão 3.30. Os parâmetros de entrada da simulação são a taxa de envio λ e o tamanho dos pacotes L de cada fluxo. O cenário é simulado para vários valores de λ e de L .

Os resultados obtidos para o sistema com prioridades são apresentados na tabela 3.4.

λ_1	L_1	λ_2	L_2		Fluxo	Valor Teórico	Valor Simulado
1	500	2	1000	L	1	0,088	[0,084; 0,086]
					2	0,305	[0,302; 0,306]
				W	1	0,088	[0,086; 0,087]
					2	0,152	[0,151; 0,152]
				L _Q	1	0,026	[0,022; 0,024]
					2	0,055	[0,052; 0,054]
				W _Q	1	0,026	[0,023; 0,023]
					2	0,027	[0,026; 0,027]
2	500	1	1000	L	1	0,156	[0,153; 0,157]
					2	0,143	[0,141; 0,144]
				W	1	0,078	[0,078; 0,078]
					2	0,143	[0,141; 0,142]
				L _Q	1	0,031	[0,029; 0,031]
					2	0,018	[0,017; 0,018]
				W _Q	1	0,016	[0,015; 0,015]
					2	0,018	[0,016; 0,017]
1	1000	2	500	L	1	0,141	[0,139; 0,143]
					2	0,161	[0,155; 0,160]
				W	1	0,141	[0,141; 0,142]
					2	0,080	[0,078; 0,079]
				L _Q	1	0,016	[0,014; 0,015]
					2	0,036	[0,031; 0,033]
				W _Q	1	0,016	[0,015; 0,015]
					2	0,018	[0,016; 0,016]

Tabela 3.4 – Simulação de um sistema M/G/1 com prioridades

Os valores teóricos são aproximadamente iguais aos valores obtidos por simulação.

3.6.4 Aproximação de Kleinrock

Para a análise da validade da aproximação de Kleinrock foram escolhidos dois cenários: um mais simples com apenas 2 ligações e 1 fluxo, apresentado na figura 3.5; e um cenário mais complexo com 6 ligações e 4 fluxos, apresentado na figura 3.6. Em ambos os cenários as ligações entre os nós têm uma capacidade igual a 64 Kbits/s e o atraso de propagação de cada ligação é de 10 ms. As filas de espera presentes têm uma capacidade de 100000 pacotes. Os valores teóricos foram calculados utilizando a equação 3.22.

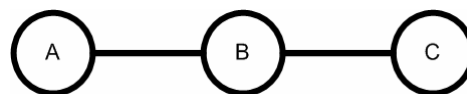


Figura 3.5 – Cenário simples para analisar a aproximação de Kleinrock

No caso do cenário da figura 3.5, é estabelecido um fluxo de pacotes de *Poisson* entre os nós A e C. O tamanho dos pacotes é exponencialmente distribuído com média 1000 *bits*. O parâmetro de entrada da simulação é a taxa de envio λ , que permite efectuar a simulação para diversos valores de ρ (0.05, 0.5, 0.95, 0.975), tendo sido obtido os resultados apresentados na tabela 3.5.

ρ	Valor Teórico (s)	Valor Simulado (s)
0.05	0,033	[0,033; 0,034]
0.5	0,063	[0,064; 0,065]
0.950	0,625	[0,380; 0,432]
0.975	1,250	[0,625; 0,783]

Tabela 3.5 – Atraso no sistema para vários valores de ρ

Dos resultados é possível concluir que a aproximação de Kleinrock é válida quando a taxa de ocupação do servidor (ρ) não é elevada. Na tabela 3.5, verifica-se que para um ρ inferior a 50 % o valor teórico é aproximadamente igual ao valor simulado; para valores elevados de ρ o valor teórico é aproximadamente o dobro do simulado e nessas situações, a aproximação de Kleinrock não é uma boa aproximação.

No cenário mais complexo, apresentado na figura 3.6, existem 4 fluxos de pacotes: CAD, ACE, BCEF e BCA, que enviam tráfego de *Poisson* às taxas 1, 4, 2 e 6 pacotes/s, respectivamente.

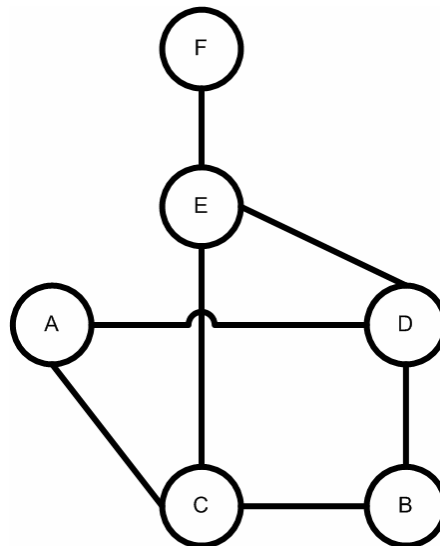


Figura 3.6 – Cenário mais complexo para analisar a aproximação de Kleinrock

A tabela 3.6 apresenta os resultados obtidos.

Fluxo	Valor Teórico (s)	Valor Simulado (s)
CAD	0,092	[0,092; 0,095]
ACE	0,094	[0,096; 0,098]
BCEF	0,135	[0,146; 0,148]
BCA	0,101	[0,105; 0,106]

Tabela 3.6 – Resultados obtidos no cenário mais complexos

Os resultados obtidos no segundo cenário (tabela 3.6), ilustram a utilidade da aproximação de Kleinrock que, mesmo num cenário mais complexo como este, representa uma forma simples e eficaz de obter importantes medidas de desempenho do sistema.

3.7 Conclusões

Este capítulo descreve os principais sistemas de filas de espera (M/M/1, M/D/1 e M/G/1) e analisa os seus parâmetros de desempenho. Os sistemas de filas de espera são classificados segundo a notação de Kendall. Os modelos apresentados neste capítulo têm vindo a ser utilizados na análise do tráfego de redes de dados, permitindo calcular analiticamente parâmetros de desempenho tais como, o atraso médio e o número médio de pacotes, no sistema e na fila de espera. A consolidação dos princípios teóricos dos sistemas de filas de espera através de cenários de simulação, permitiu comprovar o enunciado pela fórmula de Pollaczek-Khintchine, que define que o atraso médio na fila de espera de um sistema M/D/1 é metade do de um sistema M/M/1.

A validade da aproximação de Kleinrock foi analisada através de cenários de simulação. A aproximação de Kleinrock considera que cada ligação pode ser modelada por um sistema M/M/1, permitindo obter um valor aproximado do número médio e do atraso médio de pacotes. Dos resultados obtidos é possível concluir que a aproximação de Kleinrock é uma boa aproximação quando a ocupação da ligação é inferior a 50 %.

O estudo dos sistemas de filas de espera criou a necessidade de efectuar um acrescento ao NS. Um contributo desta dissertação são os dois novos objectos que foram adicionados ao código do simulador, uma fonte de tráfego Exp_Exp e um agente com capacidade de estimar medidas de desempenho do sistema. A nova fonte de tráfego foi especificada para gerar tráfego de pacotes de tamanhos exponencialmente distribuído e com intervalos de tempo entre o envio de pacotes seguindo também uma distribuição exponencial. O novo agente surge como um melhoramento de um já existente no NS, acrescentando-lhe a capacidade de

recolher o instante de envio do pacote (que permite calcular por exemplo o atraso médio dos pacotes no sistema) e corrigindo o método utilizado para calcular o número de pacotes descartados (considerando os pacotes que chegam atrasos e que não podem ser contabilizados como descartados). Os diversos cenários de simulação utilizados neste capítulo permitiram testar os novos objectos criados e validar a correcta implementação dos mesmos.

4. Algoritmos de escalonamento

Os algoritmos de escalonamento definem como diversos fluxos (serviços) interagem entre si na partilha de um meio comum, decidindo a ordem pela qual os pacotes que chegam a fila de espera de diferentes fluxos são servidos [3].

As redes IP operam segundo o paradigma *store-and-forward*, onde os pacotes são armazenados nas filas de espera dos nós e em seguida transmitidos. Portanto, é necessário perceber os mecanismos presentes nessas filas de espera, bem como o efeito que essa escolha pode ter, por exemplo quando se pretende configurar políticas de QoS numa rede. Três elementos importantes na gestão das filas de espera são o tamanho máximo da fila de espera, a técnica de descarte e o algoritmo de escalonamento presente.

A técnica de descarte, analisada no capítulo seguinte, define qual o pacote que deve ser descartado. O tamanho máximo da fila de espera tem uma limitação imposta pelo *Hardware*, ou seja, pela capacidade de armazenamento do próprio nó. Se o tamanho da fila de espera é elevado introduz-se na rede um atraso dos pacotes, atraso que pode ser suficiente para interromper aplicações e protocolos de transporte *end-to-end*: por outro lado se o tamanho da fila de espera é muito pequeno, o número de pacotes perdidos pode ser elevado. Nas secções seguintes são descritos alguns algoritmos de escalonamento, cada um projectado para resolver problemas específicos de tráfego na rede e com um efeito particular no desempenho da rede.

Este capítulo é organizado da seguinte forma. Na secção 4.1 é analisado o algoritmo de escalonamento FIFO. A secção 4.2 descreve o algoritmo de escalonamento designado FQ. O algoritmo de escalonamento SFQ é descrito na secção 4.3. Na secção 4.4 é analisado o algoritmo de escalonamento DRR. O algoritmo de escalonamento CBQ é descrito na secção 4.5. A secção 4.6 apresenta os cenários escolhidos e os resultados obtidos para o estudo dos algoritmos de escalonamento por simulação. Na secção 4.7 são apresentadas as conclusões deste capítulo.

4.1 FIFO – *First In First Out*

Esta secção tem por objectivo estudar o algoritmo de escalonamento designado FIFO. Neste algoritmo os pacotes são servidos por ordem de chegada, não sendo possível diferenciar serviços porque o fluxo de tráfego é tratado pela fila de espera como um todo.

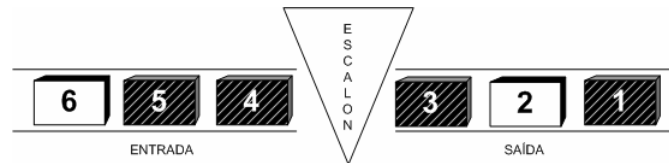


Figura 4.1 – FIFO

A figura 4.1 mostra que, à medida que os pacotes chegam, eles são colocados na fila de saída pela mesma ordem que foram recebidos. Fluxos que geram mais tráfego são mais vezes servidos, ou seja, o algoritmo de escalonamento FIFO não protege a rede de fluxos “mal comportados”. Além disso, os fluxos com pacotes maiores têm maior serviço. Este algoritmo é bastante utilizado por ser de fácil implementação.

No NS não é feita a distinção entre algoritmos de escalonamento e técnicas de descarte. Por esta razão, o algoritmo de escalonamento do tipo FIFO é designado por DropTail. O nome do algoritmo refere a técnica de descarte presente, que no caso do algoritmo FIFO tanto pode ser DropTail como DropFront (ver técnicas de descarte no capítulo 5). Assume-se a partir deste ponto que, quando se estiver a falar de algoritmos de escalonamento, o nome DropTail refere-se ao tipo FIFO, independentemente da técnica de descarte presente.

4.2 FQ – *Fair Queuing*

O algoritmo do tipo FIFO não é justo na partilha de uma ligação na presença de fluxos com débitos diferentes. A evolução natural dos algoritmos é implementar mecanismos de modo a garantir a independência dos diversos fluxos.

Nagle, em 1987 [4] propôs um mecanismo para minimizar a dependência entre fluxos, colocando cada fluxo numa fila de espera do tipo FIFO. O mecanismo é baseado num classificador que encaminha os pacotes de um determinado fluxo para a respectiva fila de espera e servindo-os segundo um algoritmo de escalonamento *Round Robin* (RR) (figura 4.2). A presença de um fluxo de débito mais elevado apenas faz com que esse fluxo aumente o tamanho da sua fila de espera, não influenciando os restantes fluxos. A largura de banda

disponível é distribuída equitativamente por todos os fluxos presentes no caso do tamanho fixo dos pacotes. O mecanismo proposto não tem em conta o tamanho dos pacotes, ou seja, fluxos de pacotes maiores têm maior serviço.

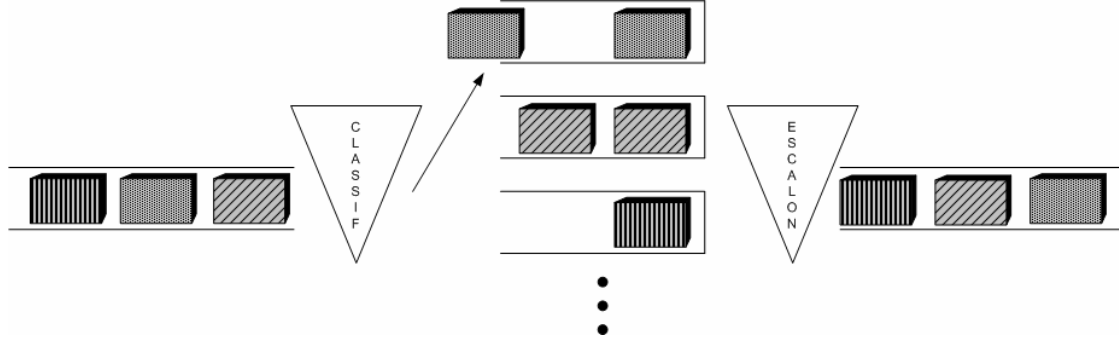


Figura 4.2 – Classificador + Escalonador (RR)

Demers, Keshav e Shenker [5] estudaram em 1990 as fraquezas do algoritmo de Nagle e propuseram o algoritmo *Fair Queuing* (FQ) que garante uma distribuição justa da largura de banda, independentemente do tamanho dos pacotes. Tal como no algoritmo de Nagle, os pacotes de cada fluxo são encaminhados para as respectivas filas de espera do tipo FIFO. Este novo algoritmo substitui a disciplina RR por um serviço *bit-a-bit round robin* (BR), onde é servido um *bit* da fila de cada vez. O serviço BR é apenas conceptual, ou seja um conceito teórico. A chegada de cada pacote é calculado um valor numérico designado por *finish number* (F_k^i), que define a ordem pela qual os pacotes são servidos. Na prática o algoritmo emula esta disciplina não enviando pacotes das filas *bit a bit*, mas sim pacote a pacote por ordem crescente de F_k^i . F_k^i representa o *finish number* do pacote k da fila do fluxo i .

O cálculo de F_k^i depende de uma variável chamada *round number* $R(t)$, que é o número de voltas dadas até o instante t num serviço BR. Considera-se $N_{ac}(t)$ como sendo o número de filas de esperas com *bits* para enviar (filas activas) e μ a taxa de transmissão da ligação a partilhar. Verifica-se na expressão

$$\partial R / \partial t = \mu / N_{ac}(t) \quad (4.1)$$

que quanto mais filas activas existem mais lento é o crescimento de $R(t)$.

Conhecido o valor de $R(t)$ é possível calcular o F_k^i de cada pacote que chega. O F_k^i de um pacote de tamanho P (em *bits*) que chega no instante t_0 a uma fila inactiva (vazia) é calculado através da expressão

$$F_k^i = R(t_0) + P \quad (4.2)$$

onde $R(t_0)$ é igual ao valor de *round number* no instante de chegada do pacote (t_0). O seu último *bit* será servido P voltas depois, no instante t .

O F_k^i de um pacote de tamanho P (em *bits*) que chega no instante t_0 a uma fila activa é calculado através da soma do *finish number* do último pacote que chegou à fila (F_{k-1}^i) com o tamanho do pacote P (em *bits*) (expressão 4.3). Por exemplo, se um pacote A de tamanho 10 *bits* chega a uma fila de espera onde já exista um pacote B com um *finish number* igual a 20, o pacote B será servido na vigésima volta e o pacote A será servido 10 voltas depois.

$$F_k^i = F_{k-1}^i + P \quad (4.3)$$

Combinando as duas expressões anteriores, obtém-se a expressão genérica para o cálculo do *finish number* de um pacote

$$F_k^i = \text{MAX}(F_{k-1}^i, R(t_0)) + P \quad (4.4)$$

Os pacotes são enviados pela mesma ordem que seriam servidos *bit a bit* numa disciplina BR, dado que $R(t)$ é uma função estritamente crescente sempre que existem pacotes para serem servidos. Quando não existem mais pacotes para serem transmitidos, o valor de $R(t)$ é igual ao valor de F_k^i calculado para o último pacote transmitido.

4.3 SFQ – *Stochastic Fair Queuing*

O algoritmo *Stochastic Fair Queuing* (SFQ), proposto em 1991 por McKenney [6], não tem uma fila de espera por cada um dos fluxos existentes, mas sim um número limitado de filas de espera do tipo FIFO pelas quais divide os fluxos existentes. O algoritmo de classificação do SFQ utiliza uma função, designada por *hash*, para mapear os diversos fluxos num número finito de filas de espera. O algoritmo de escalonamento utilizado é o RR.

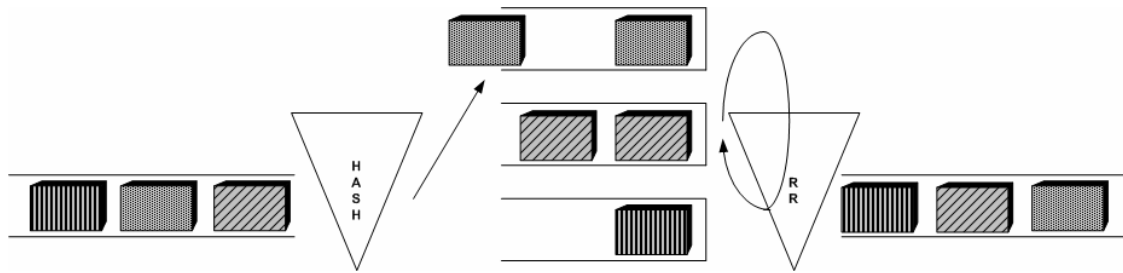


Figura 4.3 – SFQ

Comparando com o algoritmo anterior, no FQ é necessário endereçar todas as filas de espera e cada fluxo tem a sua fila de espera. O processamento do algoritmo SFQ é mais rápido porque requer menos cálculos na decisão de qual o próximo pacote a enviar, e ocupa menos recursos porque apenas as filas de espera com pacotes são endereçadas. O algoritmo usa uma lista para endereçar as filas, o que permite em cada instante saber, independentemente do número de filas activas qual a próxima fila a ser servida. Quando chega um pacote a uma fila de espera vazia, a fila de espera é colocada na lista depois da última fila servida.

O mapeamento de cada fluxo para a respectiva fila de espera é efectuado utilizando o valor calculado com a função *hash*

$$hash = ROL(src, seq) + dst \quad (4.5)$$

Os fluxos classificados com o mesmo resultado são tratados de modo idêntico, partilhando a mesma fila de espera. A função *hash* utiliza alguns campos dos pacotes, como por exemplo o endereço de origem (*src*) e o endereço de destino (*dst*), para calcular o valor do índice da fila de espera. O número sequencial (*seq*) é utilizado para alterar o resultado da função de *hash* de modo a evitar colisões repetidas de fluxos para as mesmas filas de espera, e assume um valor dentro do intervalo de 0 a 31. A função *ROL* efectua um *circular rotate-left*.

As filas são servidas seguindo um algoritmo de escalonamento RR, sem considerar o tamanho dos pacotes. Este algoritmo resolve o problema do processamento pesado do algoritmo FQ, mas tal como o do Nagle não tem em conta o tamanho do pacote. Este algoritmo descarta o pacote recebido quando o limite da respectiva fila for atingido ou o limite total for atingido.

A implementação deste algoritmo no NS tem uma lacuna: é possível num determinado cenário existirem apenas dois fluxos activos e o SFQ implementado no NS encaminhar os dois fluxos

para a mesma fila. Segundo a especificação isso não deveria acontecer, mas mesmo acontecendo, o algoritmo deveria alterar a sua função de *hash* para evitar essa situação.

4.4 DRR – *Deficit Round Robin*

O algoritmo *Deficit Round Robin* (DRR), proposto por Sheedhar e Varghese em 1995 [7], pretende distribuir uniformemente a largura de banda disponível utilizando um processo independente do comprimento dos pacotes. O DRR tenta, em cada ciclo, servir uma quantidade fixa de *bytes*, designada por limiar.

O algoritmo associa a cada fila i um crédito c_i (em *bytes*), cujo valor inicial é zero, e um limiar L_i (em *bytes*). O limiar é uma característica constante da fila; o crédito pode variar de ciclo para ciclo e só pode tomar valores não-negativos. Em cada visita à fila i o crédito disponível é incrementado de um valor igual ao limiar, ou seja, $c_i = c_i + L_i$. Nessa visita, o sistema pode servir um ou mais pacotes até esgotar o crédito disponível; um pacote só poderá ser servido num dado ciclo se houver crédito suficiente para servir todos os seus *bytes*. Quando uma fila fica vazia o crédito respectivo é colocado a zero; caso contrário, uma fila poderia estar a acumular créditos indefinidamente conduzindo eventualmente a condições de injustiça.

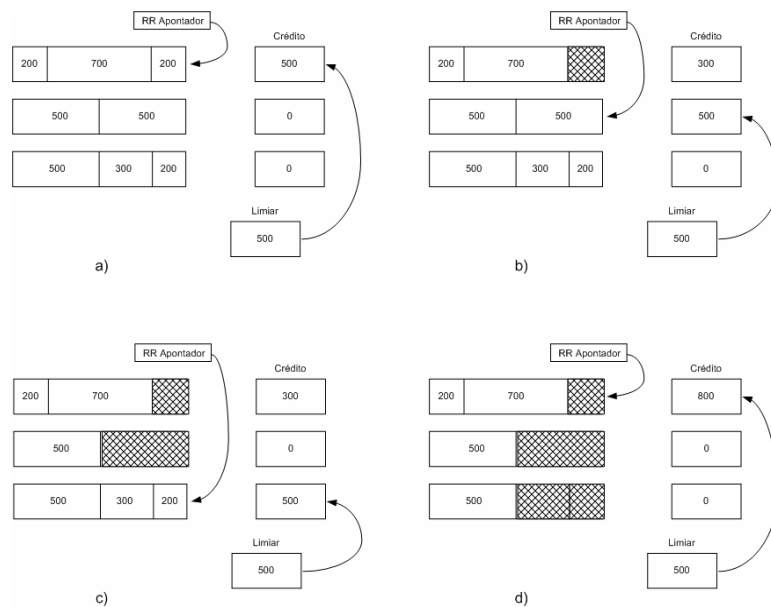


Figura 4.4 – DRR

A figura 4.4 apresenta um exemplo do princípio de funcionamento do algoritmo de escalonamento DRR. No exemplo o valor do limiar é igual a 500 *bytes*. A alínea a) representa o

instante inicial, em que é atribuído um crédito de 500 *bytes* à fila de cima. Na alínea b) o primeiro pacote da fila de cima com tamanho 200 *bytes* é transmitido e o valor do crédito da fila é actualizado para 300 *bytes*. À fila do meio é atribuído um crédito de 500 *bytes*. Na alínea c) o primeiro pacote da fila do meio com tamanho 500 *bytes* é transmitido e o valor do crédito da fila é actualizado para 0 *bytes*. À fila de baixo é atribuído um crédito de 500 *bytes*. Na alínea d) os dois primeiros pacotes com tamanho 200 e 300 *bytes* da fila de baixo são transmitidos e o valor do crédito da fila é actualizado para 0 *bytes*. À fila de cima é atribuído um crédito de 500 *bytes*, que somando com o crédito já existente de 300 *bytes*, fica com um crédito de 800 *bytes*. Se o próximo pacote a transmitir da fila de cima fosse de tamanho superior a 800 *bytes*, só seria possível transmiti-lo na próxima volta, quando se atribuisse mais um crédito no valor de 500 *bytes*.

Este algoritmo resolve o problema dos algoritmos de Nagle e de McKenney que não consideram o tamanho dos pacotes, o que pode ocasionar uma partilha “injusta” dos recursos quando existem fluxos com pacotes de tamanho diferentes. Sheedhar e Varghese escolheram o mesmo método que McKenney para mapear os fluxos num número fixo de filas de espera do tipo FIFO, utilizando uma função *hash* para o efeito. Este algoritmo de escalonamento descarta, quando necessário, o primeiro pacote da fila com mais *bytes*.

4.5 CBQ – *Class-Based Queuing*

O algoritmo *Class-Based Queuing* (CBQ), proposto por Floyd e Jacobson em 1995 [35], pretende distribuir a largura de banda disponível entre diversas classes de tráfego, tendo em conta a prioridade e a percentagem de largura de banda requerida por cada classe. O CBQ divide-se em dois módulos: um algoritmo de escalonamento tradicional que decide qual o próximo pacote a ser transmitido; e um algoritmo de escalonamento *link-sharing* que autoriza ou não o envio do pacote escolhido e garante que cada classe recebe a sua largura de banda predefinida.

No caso do algoritmo tradicional, a cada classe é associada uma fila de espera, as filas com a mesma prioridade são servidas com um algoritmo de escalonamento RR ou *weighted round-robin* (WRR). A prioridade definida para cada classe é estrita, ou seja, pacotes de maior prioridade são servidos primeiro que os pacotes de prioridade inferior.

A figura 4.5 ilustra uma configuração do CBQ possível, onde são definidas 3 classes (voz, vídeo e dados), 2 prioridades (voz e vídeo com maior prioridade) e cada classe requer uma percentagem da largura de banda disponível (voz – 3%, vídeo – 32% e dados – 65%).

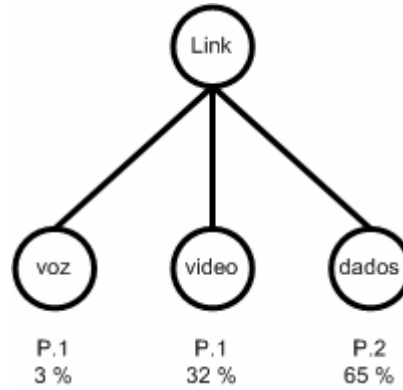


Figura 4.5 – Distribuição da capacidade da ligação

Outro exemplo é apresentado na figura 4.6, onde com o CBQ é possível criar uma estrutura hierárquica para a partilha dos recursos da ligação, utilizada por exemplo entre duas organizações A e B. À entidade A é garantida 70 % da capacidade da ligação partilhada (*Link*) e os restantes 30 % à entidade B. Em cada entidade são definidas 2 classes (voz e dados), cada uma com uma determinada prioridade e uma percentagem de largura de banda requerida.

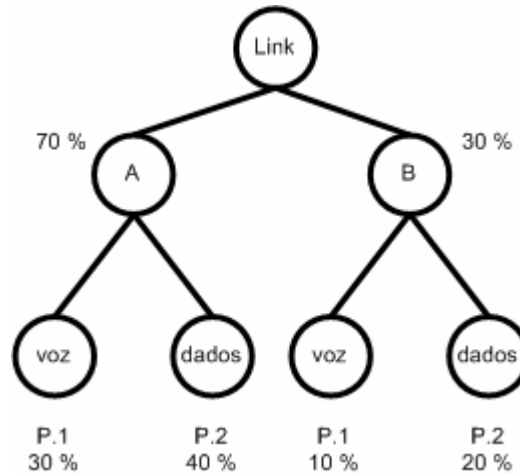


Figura 4.6 – Distribuição da capacidade da ligação com estrutura hierárquica

O algoritmo de escalonamento *link-sharing* garante a largura de banda predefinida para cada classe e redistribui a largura de banda excedente através de um método controlado. No caso da figura 4.5, se não houver dados para transmitir, os 65% da largura de banda são distribuídos entre a voz e o vídeo de acordo com os pesos de cada um. No entanto, se não houver vídeo, os 32% da largura de banda configurada para o vídeo são distribuídos apenas para a voz que tem prioridade superior aos dados. No caso da figura 4.6, se a entidade A não

tiver voz para transmitir, os 30% da largura de banda configurados para o efeito, são distribuídos para os dados da entidade A e não para a voz da entidade B.

4.6 Simulações

O objectivo desta secção é estudar por simulação os algoritmos de escalonamento FIFO, SFQ, DRR e CBQ.

As simulações efectuadas estão divididas em três partes: a primeira e a segunda comparam entre si, em dois cenários distintos, os algoritmos de escalonamento FIFO, SFQ, DRR e CBQ. A terceira simula os dois cenários (figuras 4.5 e 4.6) apresentados na especificação teórica do algoritmo CBQ.

O cenário escolhido para comparar os algoritmos de escalonamento FIFO, SFQ, DRR e CBQ é apresentado na figura 4.7.

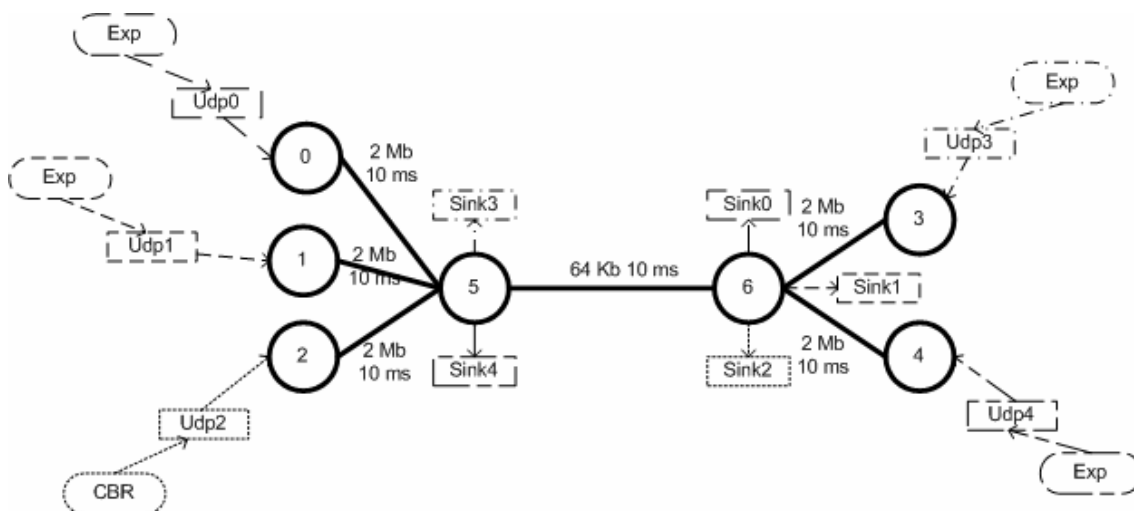


Figura 4.7 – Cenário para comparar os algoritmos de escalonamento

Na tabela 4.1 são descritos os fluxos presentes.

Fluxo	Origem -> Destino	Taxa de transmissão	Tamanho Pacotes	Sentido	Protoc. Transp.	Tipo Tráfego
Fluxo 0	Nó 0 -> Nó 6	20 kbit/s	500 bytes	5 -> 6	UDP	POISS
Fluxo 1	Nó 1 -> Nó 6	20 kbit/s -> 200 kbit/s	500 bytes	5 -> 6	UDP	POISS
Fluxo 2	Nó 2 -> Nó 6	20 kbit/s	500 bytes	5 -> 6	UDP	CBR
Fluxo 3	Nó 3 -> Nó 5	32 kbit/s	500 bytes	6 -> 5	UDP	POISS
Fluxo 4	Nó 4 -> Nó 5	32 kbit/s	50 bytes	6 -> 5	UDP	POISS

Tabela 4.1 – Características dos fluxos no cenário da figura 4.7

Cada simulação corre durante 15000 segundos. No instante 1000 segundos, o fluxo 1 altera a sua taxa de transmissão de 20 *kbit/s* (1 pacote/s) para 200 *kbit/s* (100 pacotes/s). A ligação a partilhar é bidireccional e possui uma largura de banda igual a 64*kbit/s*. O tamanho das filas de espera nos nós 5 e 6 é igual a 15 000 *bytes*. Os 2 tipos de tráfego presentes, *Poisson* e CBR, permitem analisar a influência do tipo de tráfego no desempenho dos algoritmos de escalonamento.

No cenário, o tráfego que flui em cada um dos sentidos é utilizado para simular casos diferentes. No sentido 5→6, os algoritmos são comparados na presença de um fluxo com um débito elevado. Neste cenário são analisados dois períodos da simulação: os primeiros 1000 segundos com três fluxos com o mesmo débito, e os últimos 1000 segundos com um dos fluxos de débito elevado (tempo suficiente para a alteração efectuada estabilizar). No sentido 6→5, os algoritmos são comparados na presença de fluxos com pacotes de tamanho diferente. Neste cenário são analisados os últimos 1000 segundos da simulação.

As duas próximas sub-seccões apresentam os resultados obtidos para ambos os casos.

4.6.1 Caso 1, sentido 5→6

Nesta secção é analisado o tráfego que flui no sentido 5→6 e os mecanismos de escalonamento nos dois períodos da simulação.

O cenário é simulado 10 vezes e os resultados obtidos são apresentados nas tabelas 4.2, 4.3, 4.4 e 4.5, com intervalos de confiança de 90 %.

FIFO				
Primeiros 1000,00 s	Largura de banda utilizada (%)	[93,16; 93,87]		
	Tamanho médio da Fila	Pacotes	[4,05; 4,64]	
		Bytes	[2024,57; 2320,47]	
	Largura Banda (bit/s) Descarte (%) Atraso Max. (s) Atraso Médio (s)	Fluxo 0	Fluxo 1	Fluxo 2
		[19968,96; 20184,64]	[19594,67; 19957,33]	[19888,96; 20104,64]
		[0,00; 0,01]	0.00	0.00
		[1,62; 1,83]	[1,64; 1,81]	[1,64; 1,81]
		[0,36; 0,40]	[0,37; 0,40]	[0,34; 0,37]
Últimos 1000,00 s	Largura de banda utilizada (%)	100		
	Tamanho médio da Fila	Pacotes	[28,36; 28,37]	
		Bytes	[14178,17; 14187,42]	
	Largura Banda (bit/s) Descarte (%) Atraso Max. (s) Atraso Médio (s)	Fluxo 0	Fluxo 1	Fluxo 2
		[10000,29; 10406,11]	[41016,11; 41814,29]	[12178,69; 12584,51]
		[48,37; 49,77]	[48,07; 48,95]	[35,34; 40,77]
		[1,90; 1,90]	[1,90; 1,90]	[1,89; 1,90]
		[1,86; 1,86]	[1,86; 1,86]	[1,85; 1,86]

Tabela 4.2 – FIFO, sentido 5 → 6

SFQ				
Primeiros 1000,00 s	Largura de banda utilizada (%)	[93,39; 94,07]		
	Tamanho médio da Fila	Pacotes	[3,70; 3,94]	
		Bytes	[1847,33; 1970,36]	
	Largura Banda (bit/s) Descarte (%) Atraso Max. (s) Atraso Médio (s)	Fluxo 0	Fluxo 1	Fluxo 2
		[19780,67; 20113,73]	[19825,03; 20262,97]	[19829,47; 20162,53]
		[0,54; 0,91]	[0,50; 1,04]	0,00
		[1,71; 1,78]	[1,72; 1,79]	[0,27; 0,27]
		[0,40; 0,42]	[0,39; 0,43]	[0,18; 0,19]
Últimos 1000,00 s	Largura de banda utilizada (%)	100		
	Tamanho médio da Fila	Pacotes	[11,79; 11,89]	
		Bytes	[5893,45; 5945,98]	
	Largura Banda (bit/s) Descarte (%) Atraso Max. (s) Atraso Médio (s)	Fluxo 0	Fluxo 1	Fluxo 2
		[19048,76; 19319,24]	[24680,76; 24951,24]	[19864,76; 20135,24]
		[4,09; 4,31]	[68,77; 69,31]	0,00
		[1,50; 1,53]	[1,89; 1,90]	[0,26; 0,27]
		[0,58; 0,60]	[1,47; 1,49]	[0,23; 0,24]

Tabela 4.3 – SFQ, sentido 5 → 6

DRR				
Primeiros 1000,00 s	Largura de banda utilizada (%)	[93,34; 93,73]		
	Tamanho médio da Fila	Pacotes	[4,01; 4,57]	
		Bytes	[2002,74; 2285,51]	
	Largura Banda (bit/s) Descarte (%) Atraso Max. (s) Atraso Médio (s)	Fluxo 0	Fluxo 1	Fluxo 2
		[19910,89; 20209,11]	[19741,71; 19864,69]	[19850,89; 20149,11]
		[0,00; 0,02]	[0,00; 0,04]	0,00
		[2,71; 4,11]	[2,69; 3,38]	[0,26; 0,27]
		[0,45; 0,53]	[0,43; 0,51]	[0,14; 0,21]
Últimos 1000,00 s	Largura de banda utilizada (%)	100		
	Tamanho médio da Fila	Pacotes	[29,36; 29,39]	
		Bytes	[14682,57; 14691,30]	
	Largura Banda (bit/s) Descarte (%) Atraso Max. (s) Atraso Médio (s)	Fluxo 0	Fluxo 1	Fluxo 2
		[19807,97; 20091,23]	[23908,77; 24192,03]	[19858,37; 20141,63]
		[0,61; 1,30]	[69,72; 70,19]	0,00
		[2,77; 2,81]	[2,39; 2,55]	[0,26; 0,27]
		[0,91; 1,00]	[1,33; 1,36]	[0,17; 0,18]

Tabela 4.4 – DRR, sentido 5 → 6

CBQ				
Primeiros 1000,00 s	Largura de banda utilizada (%)	[91,62; 92,18]		
	Tamanho médio da Fila	Pacotes	[4,97; 5,63]	
		Bytes	[2487,54; 2817,69]	
	Largura Banda (bit/s) Descarte (%) Atraso Max. (s) Atraso Médio (s)	Fluxo 0	Fluxo 1	Fluxo 2
		[19202,19; 19559,41]	[19325,69; 19544,71]	[19821,39; 20178,61]
		[1,59; 2,84]	[1,88; 2,51]	0,00
		[1,90; 1,94]	[1,89; 1,92]	[0,25; 0,29]
		[0,55; 0,65]	[0,57; 0,65]	0,13
Últimos 1000,00 s	Largura de banda utilizada (%)	[94,38; 95,13]		
	Tamanho médio da Fila	Pacotes	[11,51; 11,89]	
		Bytes	[5753,59; 5942,37]	
	Largura Banda (bit/s) Descarte (%) Atraso Max. (s) Atraso Médio (s)	Fluxo 0	Fluxo 1	Fluxo 2
		[19264,20; 19729,40]	[21139,58; 21154,82]	[19766,60; 20231,80]
		[1,95; 2,54]	[73,41; 73,68]	0,00
		[1,87; 1,92]	[2,01; 2,02]	0,21
		[0,61; 0,67]	1,73	0,14

Tabela 4.5 – CBQ, sentido 5 → 6

Da análise dos resultados obtidos nas tabelas 4.2 a 4.5, para os primeiros 1000 segundos da simulação, verifica-se que existe uma partilha justa da ligação entre os três fluxos. Cada fluxo transmite cerca de 20 kbit/s independentemente do algoritmo de escalonamento presente.

Quanto ao atraso máximo dos pacotes, no caso do FIFO, este é idêntico para cada fluxo, ou seja, o atraso máximo é independente do tipo de fluxo. Nos restantes casos, SFQ, DRR e CBQ, o fluxo mais regular, o CBR sofre menos atraso que os restantes. Nestes algoritmos cada fluxo tem a sua fila e não influencia o atraso máximo controlado do CBR, devido à sua taxa de transmissão constante. Como a ligação tem largura de banda suficiente para os três fluxos, em média apenas 15% da capacidade de fila de espera é utilizada e a percentagem de descarte é praticamente zero, para os 4 algoritmos. O fluxo 2 (CBR) é o mais beneficiado em qualquer um dos cenários, usufruindo de uma maior (embora ligeira) largura de banda, sem pacotes descartados e com menores atrasos.

No segundo período da simulação analisado, após a alteração imposta à simulação ter estabilizado, verifica-se que no caso do FIFO (tabela 4.2) cerca de 60 % (41 *kbit/s*) da capacidade da ligação é atribuída ao fluxo 1 (fluxo “mal comportado”) e os restantes 40 % são partilhados pelos outros dois fluxos. Esses 40 % representam para cada um, cerca de metade da capacidade utilizada nos primeiros 1000 segundos, passando de débitos perto dos 20 *kbit/s* para débitos de 10 *kbit/s* (fluxo 0) e 12 *kbit/s* (fluxo 3). No caso dos outros algoritmos, onde a cada fluxo é atribuída uma fila de espera, o fluxo “mal comportado” praticamente não influencia os fluxos 0 e 2, continuando o fluxo 2 mais beneficiado do que o fluxo 0. O atraso máximo na fila nesta situação de congestionamento é mais uma vez igual para os 3 fluxos no caso do FIFO, mas desta vez a variação desse valor é praticamente nula. Nos outros casos, o atraso máximo do fluxo CBR continua independente e aproximadamente igual nos primeiros e nos últimos 1000 segundos. O fluxo “mal comportado” apenas faz com que aumente o tamanho da sua fila de espera, a sua percentagem de descarte e o atraso máximo desse fluxo. A percentagem de descarte de pacotes do fluxo 2 incrementa de tabela em tabela, contribuindo para uma partilha mais justa entre os 3 fluxos.

Os últimos 1000 segundos de cada simulação foram analisados graficamente, para cada um dos algoritmos, obtendo os gráficos da evolução temporal da ocupação da ligação partilhada, na figura 4.8.

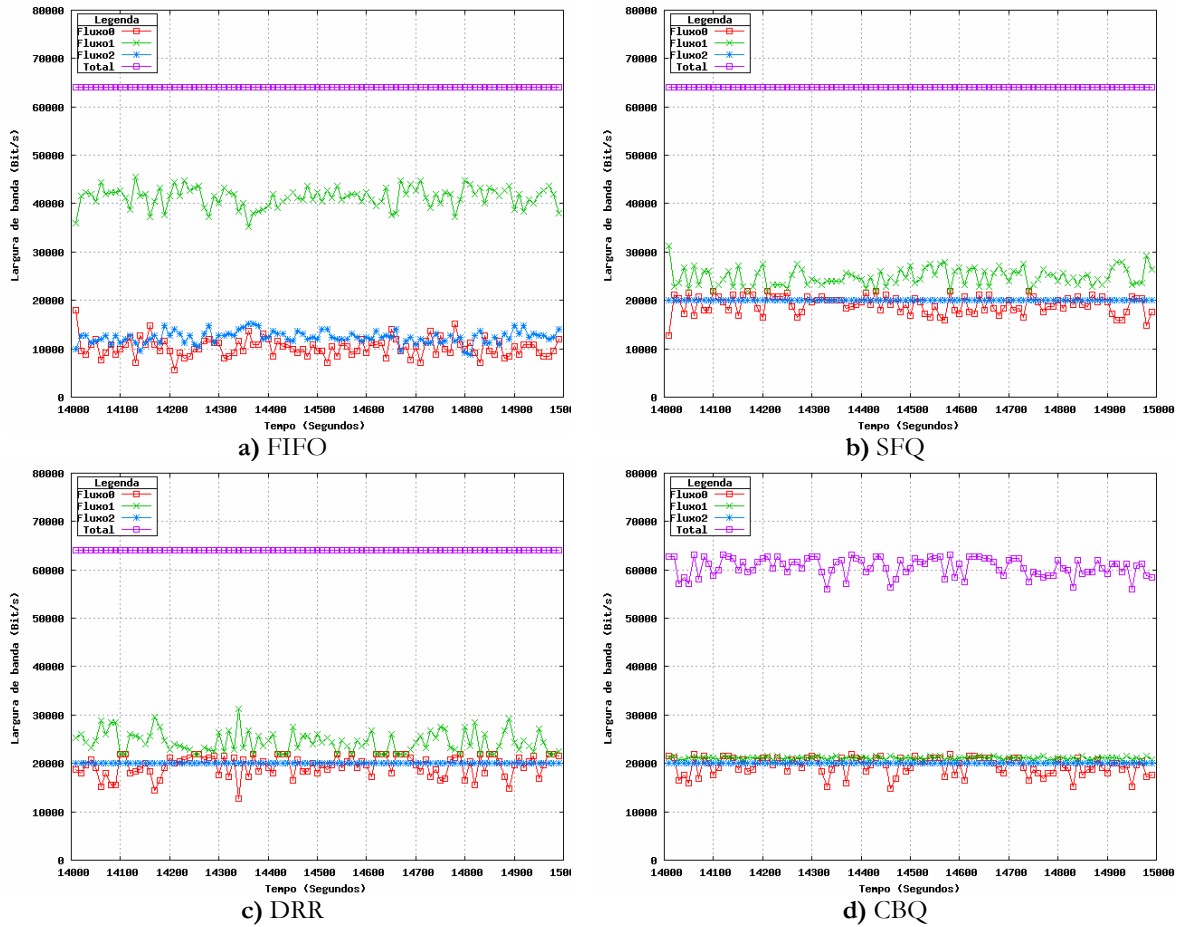


Figura 4.8 – Ocupação da ligação no sentido 5 → 6

O gráfico da figura 4.8 a) ilustra bem o comportamento do algoritmo FIFO na presença de um fluxo “mal comportado”: cerca de 40 kbit/s dos 64 kbit/s da capacidade de ligação é atribuída a esse fluxo. Nos gráficos b), c) e d) pode-se ver a independência do fluxo CBR, transmitindo a 20 kbit/s, e o fluxo 2 (o mal comportado) a transmitir a uma taxa ligeiramente superior. A figura 4.8 d) comprova os resultados obtidos na tabela 4.5, onde os 3 fluxos utilizam aproximadamente a mesma largura de banda.

4.6.2 Caso 2, sentido 6 → 5

Nesta secção é analisado o tráfego que flui no sentido 6 → 5 e os mecanismos de escalonamento nos últimos 1000 segundos.

O cenário é simulado 10 vezes e os resultados obtidos para o caso do tamanho fixo dos pacotes são apresentados nas tabelas 4.6 a 4.9, com intervalos de confiança de 90 %.

FIFO			
Últimos 1000,00 s	Largura de banda utilizada (%)	[95,84; 96,45]	
	Tamanho médio da Fila	Pacotes	[14,68; 15,52]
		Bytes	[1309,93; 1392,15]
	Largura Banda (<i>bit/s</i>) Descarte (%) Atraso Max. (s) Atraso Médio (s)	Fluxo 3	Fluxo 4
		[29905,24; 30413,16]	[31120,04; 31627,96]
		[5,57; 6,27]	[5,47; 6,14]
		[0,70; 0,75]	[0,67; 0,73]
		[0,25; 0,27]	[0,19; 0,21]

Tabela 4.6 – FIFO, no sentido 6 → 5

SFQ			
Últimos 1000,00 s	Largura de banda utilizada (%)	[89,24; 89,61]	
	Tamanho médio da Fila	Pacotes	[5,43; 5,53]
		Bytes	[431,95; 446,14]
	Largura Banda (<i>bit/s</i>) Descarte (%) Atraso Max. (s) Atraso Médio (s)	Fluxo 3	Fluxo 4
		[31560,07; 31983,93]	[25248,47; 25672,33]
		[0,00; 0,04]	[23,47; 24,15]
		[0,47; 0,55]	[0,71; 0,72]
		0,13	[0,10; 0,11]

Tabela 4.7 – SFQ, no sentido 6 → 5

DRR			
Últimos 1000,00 s	Largura de banda utilizada (%)	[99,65; 99,89]	
	Tamanho médio da Fila	Pacotes	[116,48; 134,87]
		Bytes	[9433,90; 10297,01]
	Largura Banda (<i>bit/s</i>) Descarte (%) Atraso Max. (s) Atraso Médio (s)	Fluxo 3	Fluxo 4
		[30961,99; 31166,01]	[32686,59; 32890,61]
		[2,28; 3,38]	[1,19; 1,63]
		[2,94; 3,33]	[2,74; 2,98]
		[1,03; 1,11]	[1,32; 1,55]

Tabela 4.8 – DRR, no sentido 6 → 5

CBQ			
Últimos 1000,00 s	Largura de banda utilizada (%)	[94,72; 95,56]	
	Tamanho médio da Fila	Pacotes	[10,30; 11,10]
		Bytes	[1516,33; 1786,64]
	Largura Banda (<i>bit/s</i>) Descarte (%) Atraso Max. (s) Atraso Médio (s)	Fluxo 3	Fluxo 4
		[31572,21; 32170,19]	[28720,61; 29318,59]
		[0,38; 0,52]	[10,09; 24,54]
		[1,65; 1,66]	[0,73; 0,81]
		[0,36; 0,42]	0,14

Tabela 4.9 – CBQ, no sentido 6 → 5

A percentagem de descarte no caso do FIFO é idêntica para os dois fluxos (tabela 4.6). Intuitivamente poder-se-ia pensar que o fluxo 4 que envia 10 vezes mais pacotes que o 3, tivesse uma percentagem de descarte mais elevada. Isso não acontece porque esse fluxo é mais vezes servido que o fluxo 3, o que equilibra a contagem final. Os algoritmos SFQ e o CBQ (ambos utilizam o mesmo princípio RR e não têm em conta o tamanho dos pacotes) beneficiam o fluxo 3 descartando menos pacotes deste fluxo. O DRR é o algoritmo que melhor se adapta a este cenário, oferecendo a cada fluxo cerca de metade da capacidade da ligação, independentemente do tamanho dos seus pacotes.

Os últimos 1000 segundos de cada simulação foram analisados graficamente, para cada um dos algoritmos, obtendo os gráficos da evolução temporal da ocupação da ligação partilhada, na figura 4.9.

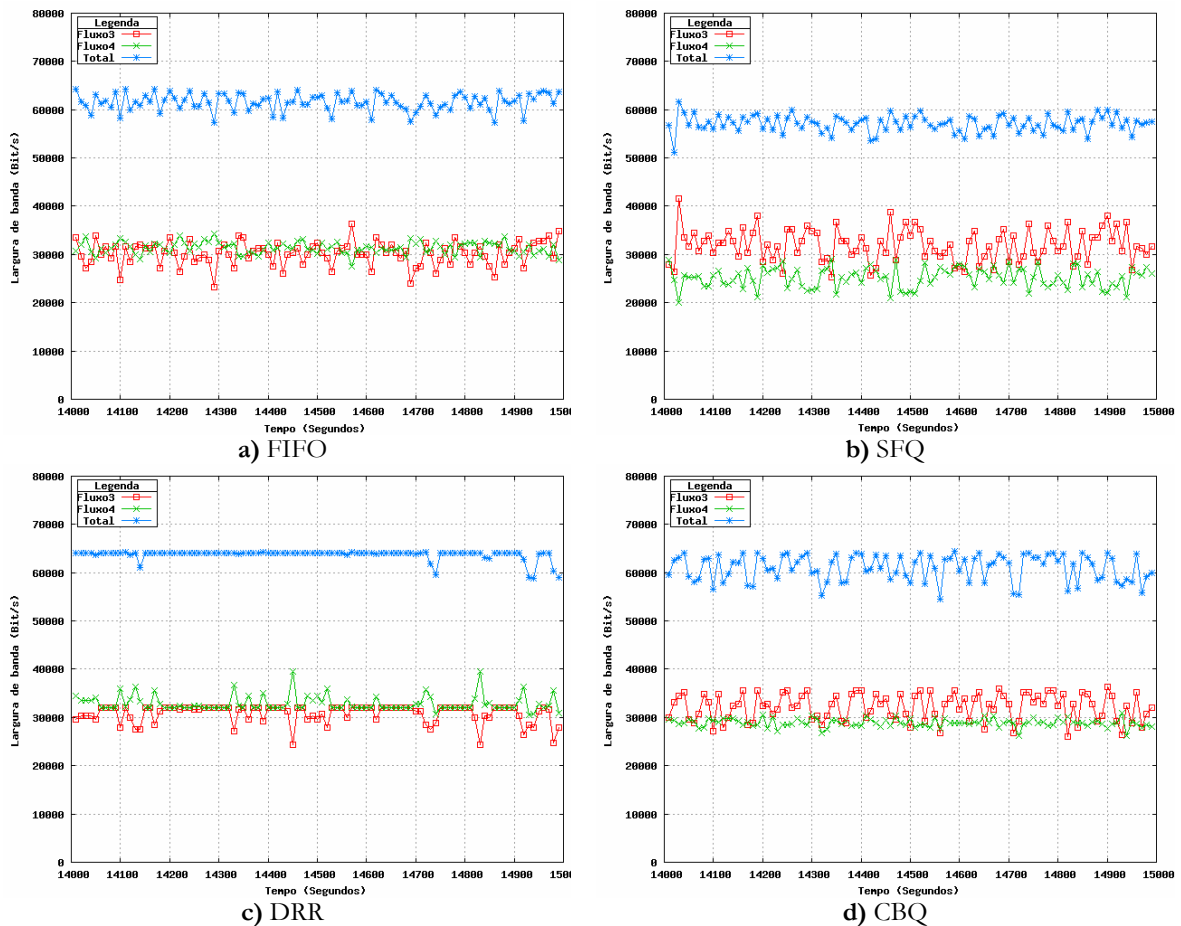


Figura 4.9 – Ocupação da ligação no sentido 6 → 5

A figura 4.9 a) ilustra a partilha justa da ligação independentemente do tamanho dos pacotes de cada fluxo. Da análise dos gráficos b) e d) verifica-se que o fluxo 4 (o que tem pacotes de tamanho mais pequenos) é prejudicado na partilha da ligação quando ambos transmitem à mesma taxa. O SFQ e o CBQ não têm em conta o tamanho do pacote, prejudicando o fluxo com pacotes mais pequenos. O DRR (figura 4.9 c)) é o que melhor se adapta a esta situação, porque considera o tamanho dos pacotes, partilhando entre os 2 fluxos a largura de banda disponível equitativamente. Este algoritmo permite obter um melhor aproveitamento da largura de banda disponível.

4.6.3 CBQ

Nesta secção são estudadas as especificidades do CBQ. O cenário escolhido para simular o desempenho do CBQ considerando o primeiro exemplo (figura 4.5) é apresentado na figura 4.10.

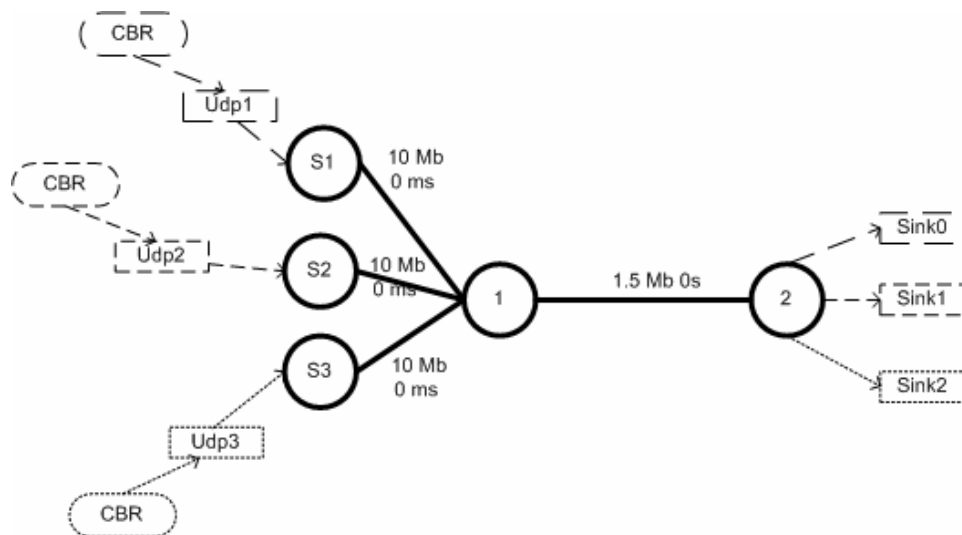


Figura 4.10 – Cenário para o CBQ

Na tabela 4.10 são descritos os fluxos presentes.

Fluxo	Origem → Destino	Taxa de transmissão	Prioridade	% LB	Tamanho Pacotes	Protoc. Transp.	Tipo Tráfego
Vídeo	Nó S1 → Nó 2	1.52 Mbit/s	1	32	190 bytes	UDP	CBR
Áudio	Nó S2 → Nó 2	2 Mbit/s	1	3	500 bytes	UDP	CBR
Dados	Nó S3 → Nó 2	1.6 Mbit/s	2	65	1000 bytes	UDP	CBR

Tabela 4.10 – Características dos fluxos no cenário

A simulação corre durante 40 segundos. Neste cenário são definidas duas prioridades: uma prioridade alta (1) para o vídeo e para o áudio, e uma prioridade baixa (2) para os dados. A ligação a partilhar é bidireccional (entre o nó 1 e nó 2) possui uma largura de banda igual a 1.5

Mbit/s. Os nós S1, S2 e S3 estão ligados ao nó 1 com uma largura de banda de 10 Mbit/s. O tamanho das filas de espera no nó 1 é igual a 20 pacotes (uma fila de espera por fluxo). O algoritmo de escalonamento escolhido no CBQ é o WRR, configurado para atribuir 32% ao fluxo de dados, 3% ao fluxo de áudio e 32% ao fluxo de vídeo. No instante 4, o fluxo de dados é desligado durante 4 segundos. No instante 12, o fluxo de áudio é desligado durante 6 segundos. No instante 20, o fluxo de vídeo é desligado durante 4 segundos.

A figura 4.11 apresenta a evolução temporal em percentagem da ocupação da ligação partilhada.

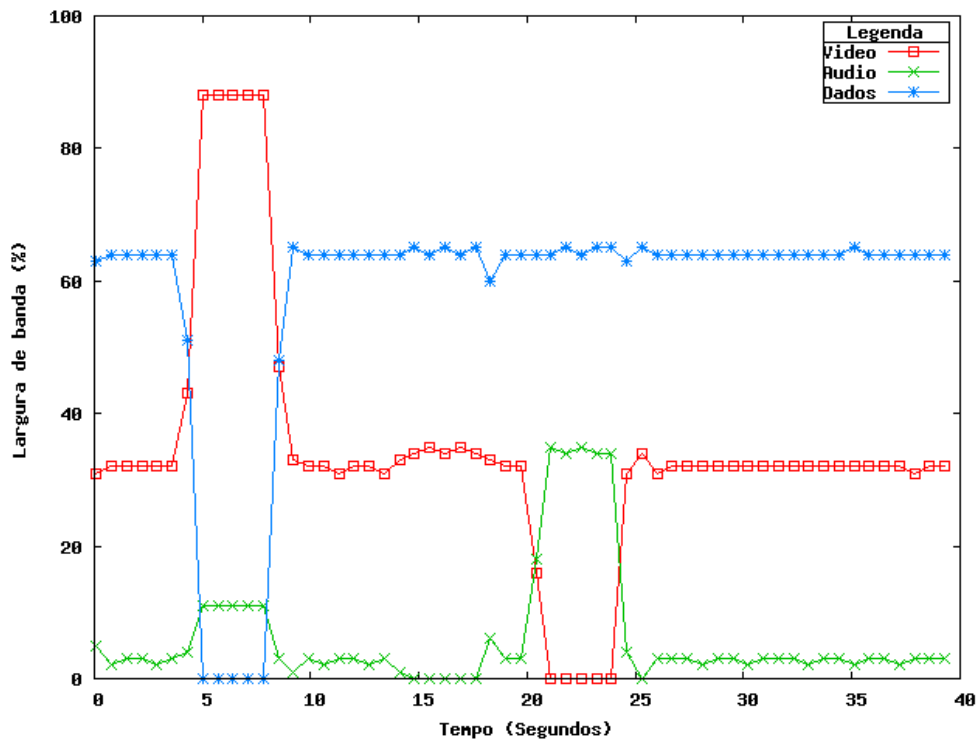


Figura 4.11 – CBQ simples

Da análise do primeiro cenário verifica-se, na figura 4.11, que nos 4 segundos iniciais cada fluxo ocupa a largura de banda predefinida (3% para o áudio, 32% para o vídeo e 65% para os dados). Quando o fluxo de dados é interrompido no instante 4, os 65% da largura de banda libertados são divididos pelo fluxo de vídeo e o de áudio em regularidade com as percentagens de largura de banda configuradas, numa relação de 10 para 1. Se um dos fluxos de maior prioridade é desligado (instante 12 ou 20), a largura de banda libertada é atribuída apenas ao outro fluxo com a mesma prioridade.

A figura 4.12 apresenta o cenário escolhido para simular o segundo exemplo, descrito na figura 4.6 da secção 4.5, para o caso do CBQ hierárquico.

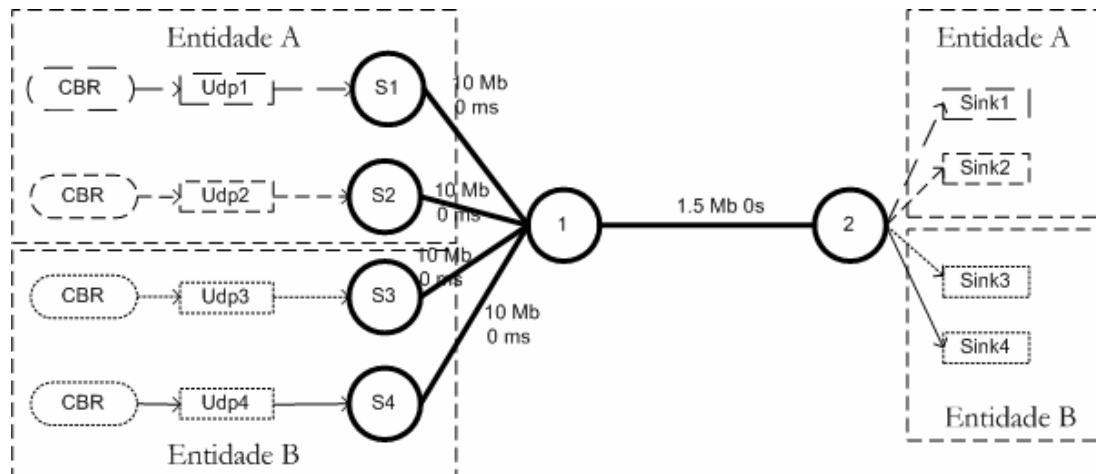


Figura 4.12 – Cenário para o CBQ hierárquico

Na tabela 4.11 são descritos os fluxos presentes.

Fluxo	Origem → Destino	Taxa de transmissão	Prioridade	% LB	Tamanho Pacotes	Protoc. Transp.	Tipo Tráfego
VídeoA	Nó S1 → Nó 2	1.52 Mbit/s	1	30	190 bytes	UDP	CBR
DadosA	Nó S2 → Nó 2	1.6 Mbit/s	2	40	1000 bytes	UDP	CBR
VídeoB	Nó S3 → Nó 2	2 Mbit/s	1	10	500 bytes	UDP	CBR
DadosB	Nó S4 → Nó 2	1.6 Mbit/s	2	20	1000 bytes	UDP	CBR

Tabela 4.11 – Características dos fluxos no cenário da figura 4.15

A simulação corre durante 40 segundos. Duas prioridades são definidas: uma prioridade alta (1) para o vídeo e uma prioridade baixa (2) para os dados. A ligação a partilhar é bidireccional, possui uma largura de banda igual a 1.5 Mbit/s. O tamanho das filas de espera no nó 1 é igual a 20 pacotes (uma fila de espera por fluxo). O algoritmo de escalonamento escolhido no CBQ é o WRR. A largura de banda é dividida entre as duas entidades, 70% para a entidade A e 30% para a entidade B. Na entidade A, 30% da largura de banda é para o fluxo de vídeo e 40% para o fluxo de dados. Na entidade B, 10% da largura de banda é para o fluxo de vídeo e 20% para o fluxo de dados. No instante 12, o fluxo de vídeo da entidade A é desligado durante 4 segundos. No instante 28, o fluxo de dados da entidade B é desligado durante 4 segundos.

A figura 4.13 apresenta a evolução temporal da ocupação da ligação partilhada em percentagem.

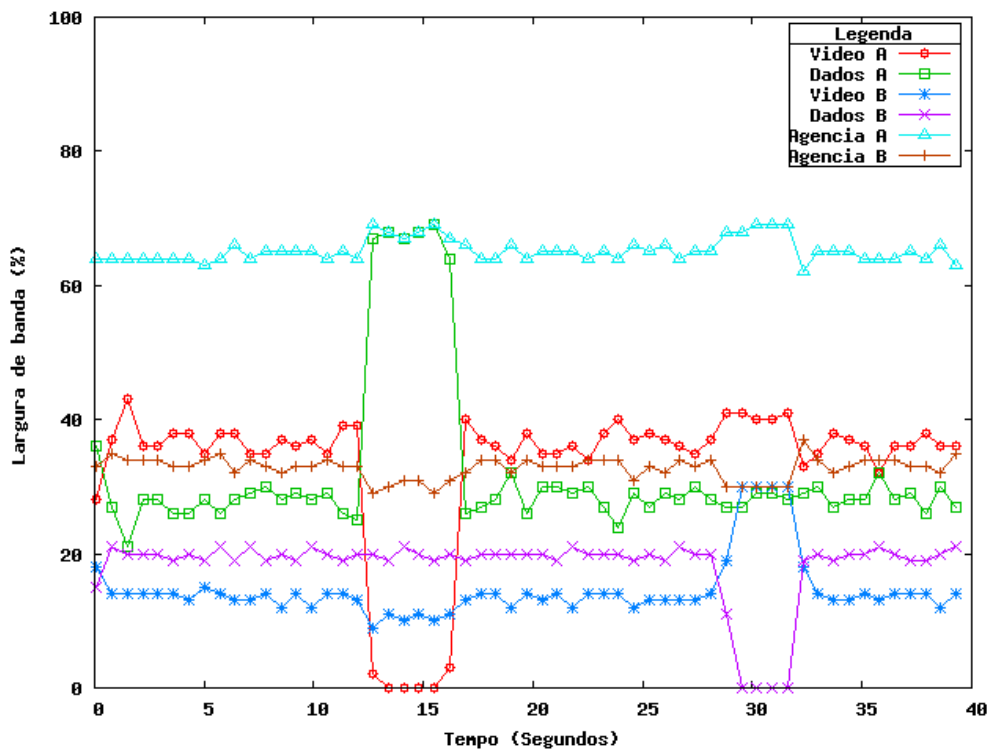


Figura 4.13 – CBQ hierárquico

No caso do resultado obtido para o CBQ hierárquico (figura 4.13), a largura de banda libertada quando um dos fluxos de uma das entidades é desligado, é atribuída apenas ao outro fluxo da entidade, independentemente de existir na outra entidade um fluxo de maior prioridade. Por exemplo, no instante 12, quando o fluxo de vídeo da entidade A é desligado, apenas o fluxo de dados da entidade A aumenta a sua percentagem de ocupação da largura de banda para o valor máximo da entidade (70%).

Em ambos os cenários a largura de banda libertada não é distribuída aleatoriamente, mas sim respeitando configurações predefinidas.

4.7 Conclusões

Este capítulo permitiu analisar, simular e comparar os algoritmos de escalonamento FIFO, SFQ, DRR e CBQ.

O FIFO é o algoritmo mais simples; os seus pacotes são simplesmente servidos por ordem de chegada e os fluxos de tráfego que cheguem são tratados pela fila de espera como um todo. O SFQ possui um conjunto limitado de filas de espera do tipo FIFO pelas quais divide fluxos de tráfego que cheguem, enviando um pacote de cada fila seguindo um algoritmo RR. Este

algoritmo não tem em conta o tamanho dos pacotes. O DRR também possui um conjunto limitado de filas de espera do tipo FIFO pelas quais divide fluxos de tráfego que cheguem, tendo como objectivo distribuir uniformemente a largura de banda utilizando um processo independente do tamanho dos pacotes. O DRR tenta, em cada ciclo, servir uma quantidade fixa de *bytes*, designada por limiar. Por último, o CBQ é um algoritmo mais complexo; este também possui um conjunto limitado de filas de espera do tipo FIFO pelas quais divide fluxos de tráfego que cheguem, distribuindo a largura de banda disponível tendo em conta a prioridade e a percentagem de largura de banda requerida por cada fluxo. O CBQ divide-se em dois módulos: um algoritmo de escalonamento tradicional (RR ou WRR) que decide qual o próximo pacote a ser transmitido; e um algoritmo de escalonamento *link-sharing* que autoriza ou não, o envio do pacote escolhido e garante que cada classe recebe a sua largura de banda predefinida.

Através de um mesmo cenário foi possível comparar os algoritmos de escalonamento em duas situações distintas, analisando cada sentido de envio de tráfego. Num dos sentidos os algoritmos são comparados na presença de 3 fluxos, em que um deles é “mal comportado” (com uma taxa de transmissão elevada) No caso dos algoritmos SFQ, DRR e CBQ, como a cada fluxo é associada uma fila, esse fluxo faz com que a sua fila de espera atribuída aumente, e aumente a percentagem de descarte de pacotes e o atraso máximo desse fluxo, não afectando os restantes fluxos. No caso do algoritmo FIFO esse fluxo efecta os restantes fluxos. O CBQ é o algoritmo que controla com mais rigor o fluxo “mal comportado”, descartando mais pacotes deste fluxo, permitindo uma partilha entre os 3 fluxos mais justa. No outro sentido e na presença de 2 fluxos com tamanhos de pacotes distintos transmitindo à mesma taxa, no caso do algoritmo FIFO, como os fluxos transmitem à mesma taxa eles partilham equitativamente a largura de banda disponível. Os algoritmos SFQ e CBQ, como não têm em conta o tamanho dos pacotes, não partilham equitativamente a largura de banda e descartam mais pacotes do fluxo com pacotes pequenos. Dos quatro algoritmos estudados, o DRR é o que mais se adapta às diversas situações simuladas, por ter em conta o tamanho dos pacotes.

A última parte das simulações permitiu consolidar os princípios de funcionamento enunciados na apresentação do algoritmo de escalonamento CBQ. A largura de banda é dividida entre os fluxos tendo em conta a prioridade e a largura de banda requerida por cada fluxo. A distribuição da largura de banda extra, que por exemplo existe quando um dos fluxos deixa de

transmitir libertando os recursos utilizados, nunca é aleatória. Essa distribuição tem em conta a estrutura configurada, hierárquica ou não, e é controlada pelo algoritmo *link-sharing* presente.

5. Técnicas de descarte

As técnicas de descarte definem quais os pacotes, numa fila de espera de um *router*, que devem ser descartados em determinado instante. Este descarte pode ser efectuado numa situação de congestionamento ou com o objectivo de evitar essa situação [3].

As técnicas de descarte analisadas neste capítulo são a DropTail, a DropFront e a RED (*Random Early Detection*). As duas primeiras descartam pacotes das suas filas de espera sempre que não têm espaço disponível para receber o pacote que acaba de chegar. Quando isso acontece a técnica de descarte DropTail descarta o pacote que acaba de chegar, e a técnica de descarte DropFront descarta o primeiro pacote da fila de espera, ou seja o próximo pacote que iria ser transmitido. No caso da técnica de descarte RED são descartados aleatoriamente, com uma determinada probabilidade dependente do tamanho da fila de espera, os pacotes que chegam, com o objectivo de manter o tamanho médio da fila de espera entre os valores configurados.

O propósito deste capítulo é estudar por simulação as três técnicas de descarte de pacotes, analisando as principais diferenças entre elas e compreendendo o princípio de funcionamento de cada uma delas. Essas técnicas diferem entre si segundo dois aspectos: na escolha do pacote da fila de espera a descartar e/ou na condição para efectuar esse descarte. No que diz respeito ao primeiro aspecto, são comparadas por simulação as técnicas de descarte DropTail e DropFront, analisando o efeito da escolha do pacote a descartar nos parâmetros de desempenho da rede, tais como o atraso máximo na fila e o número de pacotes descartados. O segundo aspecto é analisado comparando por simulação a técnica de descarte DropTail com a técnica de descarte RED, verificando a mais valia introduzida pela técnica de descarte RED no desempenho do controlo de congestionamento do protocolo de transporte TCP.

Este capítulo é organizado da seguinte forma. Na secção 5.1 é analisada a escolha do pacote a descartar e são descritas as condições para o descarte de pacotes numa fila de espera. Na secção 5.2 descreve-se analiticamente a técnica de descarte RED. A secção 5.3 apresenta os cenários escolhidos e os resultados obtidos por simulação, para o estudo das três técnicas de descarte. Na secção 5.4 são apresentadas as conclusões deste capítulo.

5.1 Posição do pacote a descartar e condição de descarte

Como já foi referido, as técnicas de descarte distinguem-se pela posição na fila de espera do pacote a descartar e pela condição que define o descarte dos pacotes.

A situação mais comum e a mais simples de implementar é o descarte do último pacote recebido. Neste caso, quando um pacote chega à fila de espera, verifica-se se a fila tem espaço disponível para receber o pacote: se não tiver descarta o pacote. O descarte do primeiro pacote da fila é mais complexo porque implica alterar o ponteiro que indica qual o próximo pacote a transmitir. Além disso, no caso dos recursos da fila de espera serem geridos em *bytes*, é necessário verificar se o tamanho do pacote descartado é igual ou superior ao tamanho do pacote que acaba de chegar; se assim não for, é necessário descartar um número de pacotes necessário para haver capacidade na fila para receber o novo pacote. Escolher um pacote na fila de espera aleatoriamente para descartar é ainda mais complexo, pois requer a existência de um ponteiro para cada pacote, o que torna mais complexa a gestão da fila de espera. Na prática, devido à sua complexidade, esta técnica não é regularmente implementada.

Tradicionalmente é necessário descartar pacotes quando os recursos da fila de espera estão esgotados, utilizando uma técnica do tipo DropTail ou DropFront. No entanto, para evitar uma situação de congestionamento da rede, pode-se iniciar o descarte de pacotes quando a fila de espera atinge um determinado limiar. Este processo é utilizado pela técnica RED. O RED será descrito com maior detalhe na secção seguinte.

5.2 RED – *Random Early Detection*

Este algoritmo foi desenvolvido por Sally Floyd e Van Jacobson em 1990 [8]. Ao contrário das técnicas de descarte tradicionais que descartam os pacotes quando a fila está cheia, a técnica de descarte designada por RED descarta com uma certa probabilidade os pacotes que chegam. A probabilidade de descartar os pacotes aumenta com o crescimento do tamanho médio da fila de espera. Se, durante um intervalo de tempo, a fila de espera se encontra com poucos pacotes, nenhum pacote é descartado; por outro lado, se o número de pacotes na fila de espera excede um determinado limiar inferior, os pacotes são descartados com uma certa probabilidade; se o número de pacotes exceder um determinado limiar superior, todos os pacotes serão descartados.

Além do descarte, também existe a possibilidade de marcar o pacote afectando o campo ECN (*Explicit Congestion Notification*) do seu cabeçalho, e deste modo informar a fonte que a ligação está a ficar congestionada e que deve baixar a taxa de transmissão. Este processo é normalmente utilizado em conjunto com o protocolo de transporte TCP, pelo facto de este permitir o controlo da taxa de transmissão da fonte.

O algoritmo RED consiste em duas fases: a primeira em que se estima o tamanho médio da fila de espera, e a segunda em que se decide se se deve descartar ou não o pacote que chega.

O tamanho médio da fila de espera ($TamMed$) é calculado através de uma leitura periódica das amostras do valor do tamanho da fila de espera ($AmTam$) e do valor do tamanho médio da fila de espera no cálculo anterior e é dado por

$$TamMed = (1 - \alpha) \times TamMed + \alpha \times AmTam \quad (5.1)$$

onde α é um valor entre 0 e 1.

Para decidir se se deve ou não descartar o pacote que chega, são utilizados 2 parâmetros: um limiar superior e um inferior ($MaxThresh$ e $MinThresh$). O limiar inferior define o valor médio de pacotes na fila de espera abaixo do qual nenhum pacote que chegue é descartado; o limiar superior define o valor médio de pacotes na fila de espera acima do qual todos os pacotes que cheguem são descartados. Entre os dois limiares, os pacotes são descartados com uma probabilidade (pa) que é calculada em função do valor médio da fila de espera ($TamMed$). A relação entre as duas grandezas é apresentada na figura 5.1.

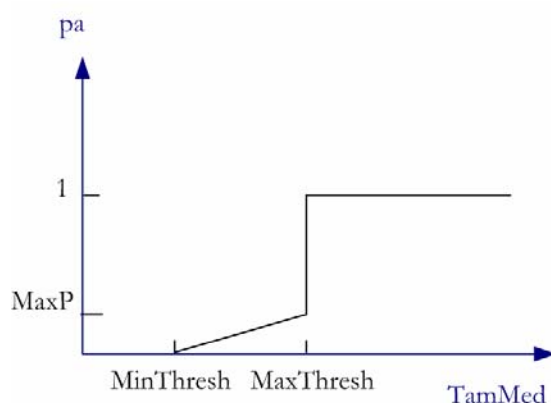


Figura 5.1 – Probabilidade de descarte em função do tamanho médio da fila

A probabilidade pa aumenta com $TamMed$, quando este se encontra entre $MinThresh$ e $MaxThresh$. O tamanho máximo de pa para este intervalo é $MaxP$. Nesse instante, o valor da

probabilidade é igual à unidade e começam a ser descartados todos os pacotes que chegam à fila de espera.

A figura traduz a ideia de que pa é calculado apenas em função de $TamMed$, mas na realidade é um pouco mais complexo. A probabilidade de descarte é calculada em função de $TamMed$ e em função do intervalo de tempo desde o descarte do último pacote e é dada por

$$pa = TempP / (1 - count \times TempP) \quad (5.2)$$

O valor $count$ da expressão 5.2 representa o número de pacotes que não foram descartados enquanto o tamanho médio da fila de espera se encontra entre os dois limiares. O valor $TempP$ representa o segmento de recta entre $MinThresh$ e $MaxThresh$ da figura 5.1 e é calculado através da expressão

$$TempP = MaxP \times (TamMed - MinThresh) / (MaxThresh - MinThresh) \quad (5.3)$$

A probabilidade de descarte aumenta lentamente à medida que o valor $count$ é incrementado, fazendo com que os pacotes descartados não sejam consecutivos, mas sim espaçados no tempo. Em situações em que os pacotes de uma determinada ligação vêm agrupados (uma rajada de pacotes), se não existisse este mecanismo, mais que um pacote desse conjunto seria descartado, quando apenas é necessário descartar um pacote para notificar a fonte que deve reduzir a taxa de transmissão.

A técnica de destarte RED mantém o tamanho da fila de espera reduzido, permitindo aos diversos fluxos TCP uma adaptação gradual às condições da rede, evitando situações de congestionamento. Este algoritmo foi desenhado para ser utilizado com o protocolo de transporte TCP, de modo a melhorar o seu desempenho e evitar a sincronização global. Este efeito de sincronização global acontece quando, numa situação de congestionamento, pacotes de diversas fontes são descartados, e consequentemente, várias fontes entram em *Slow Start* simultaneamente. Nesta situação, a redução e o aumento da taxa de envio das fontes é sincronizado, o que faz com que a largura de banda utilizada pelo conjunto das fontes não seja constante. Com a utilização do RED, alguns pacotes começam a ser descartados antes da situação de congestionamento ocorrer e, assim sendo, apenas algumas fontes necessitam de reduzir a sua taxa de transmissão para evitar uma situação de congestionamento.

Na arquitectura DiffServ (*Differentiated Services*), que será descrita e estudada no capítulo 8, o IETF (*Internet Engineering Task Force*) definiu uma variante do RED como técnica de implementação dos 3 níveis de descarte presentes em cada classe do AF PHB (*Assured Forwarding Per-Hop Behavior*). Normalmente cada nível de descarte é designado por uma cor: verde (nível baixo de descarte), amarelo (nível médio de descarte) e vermelho (nível elevado de descarte). Cada fila de espera é construída por 3 filas de espera virtuais, onde cada uma das filas virtuais é configurada com os parâmetros da técnica de descarte RED, de modo a simular os 3 níveis de descarte distintos. Existem algumas variantes do RED descritas na literatura: GRED (*Generalized RED*), WRED (*Weighted RED*), RIO-C (*RED with In/Out and Coupled virtual Queues*) e RIO-DC (*RED with In/Out and De-Coupled Queues*). As variantes diferem no cálculo do tamanho das respectivas filas de esperas virtuais. No GRED, que consiste em 16 filas de espera independentes do tipo RED, o tamanho médio de cada fila é independente e é calculado tendo em conta somente os pacotes da própria fila. O GRED permite definir prioridades entre filas. No WRED o tamanho médio de cada uma das filas virtuais é igual e engloba no seu cálculo os pacotes das 3 filas de espera virtuais. No RIO-C o tamanho médio de cada fila virtual é calculado tendo em conta os pacotes da própria fila e das filas virtuais de menor nível de descarte. No RIO-DC o tamanho médio de cada fila virtual é calculado tendo em conta somente os pacotes da própria fila virtual. Um estudo comparativo destas variantes é apresentado em [32].

5.3 Simulações

O objectivo desta secção é estudar por simulação as três técnicas de descarte de pacotes. As simulações efectuadas estão divididas em três secções. A primeira secção compara as técnicas de descarte DropTail e DropFront, avaliando o efeito da escolha do pacote a descartar nos parâmetros de desempenho da rede para o caso dos protocolos de transporte UDP e TCP. A segunda secção analisa os parâmetros configuráveis da técnica de descarte RED, esclarecendo o princípio de funcionamento desta técnica de descarte. Por último é analisada a condição de descarte de pacotes, comparando a técnica de descarte DropTail com a técnica de descarte RED, verificando a mais valia introduzida pela técnica de descarte RED no desempenho do controlo de congestionamento do protocolo de transporte TCP.

O cenário escolhido é apresentado na figura 5.2. O objectivo das simulações é analisar a ligação entre os nós R1 e R2. A fila de espera do nó R1 é de tamanho igual a 20 pacotes. Cada simulação tem a duração de 20 segundos.

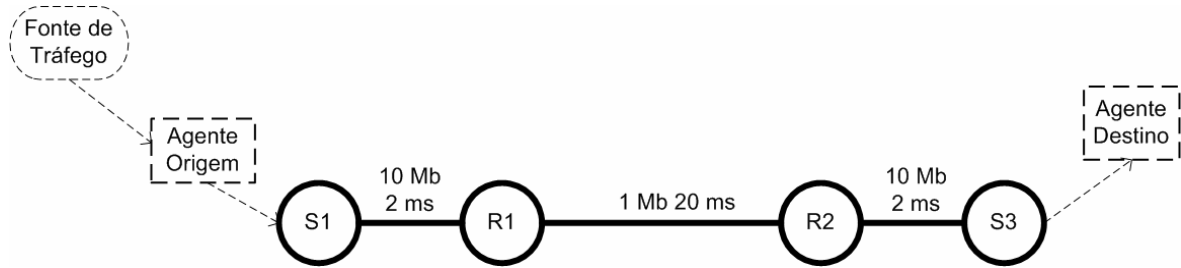


Figura 5.2 – Cenário escolhido para comparar as técnicas de descarte

5.3.1 DropTail Vs DropFront - Posição do pacote a descartar

Nesta secção são comparadas as técnicas de descarte DropTail e DropFront utilizando primeiro o protocolo de transporte TCP e de seguida o UDP.

TCP

Para o caso do TCP, o agente de origem utilizado é do tipo Reno com uma janela de tamanho máximo igual a 50 pacotes, simulando a transferência de 2 *Mbytes* de dados via FTP. Os resultados obtidos são apresentados na tabela 5.1.

	Pacotes Enviados	Pacotes Retransmitidos	ACK Recebidos	LB (Kbits/s)	Fim de dados na fila (s)
DropTail	2054	53	2025	912.490	18.46
DropFront	2022	21	2009	905.277	18.46

Tabela 5.1 – Resultados obtidos com TCP, DropTail Vs DropFront

Os resultados da tabela 5.1 mostram que o número de pacotes retransmitidos no caso da técnica de descarte DropFront é aproximadamente metade do que no caso da técnica de descarte DropTail. A análise das janelas de congestionamento e da evolução do descarte de pacotes ao longo do tempo, apresentadas nas figuras 5.3 e 5.4, respectivamente, permite explicar a razão dessa diferença.

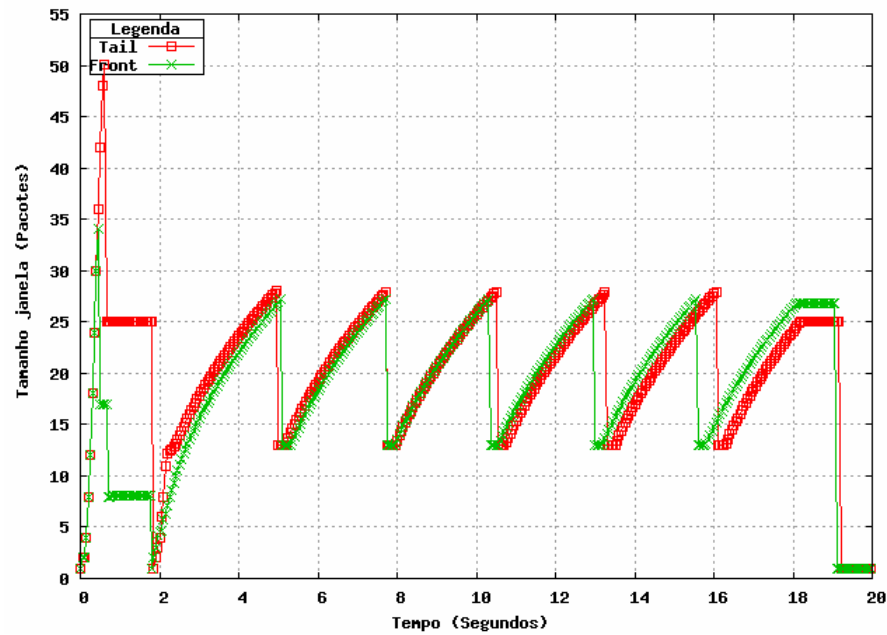


Figura 5.3 – Janela de Congestionamento, DropTail Vs DropFront

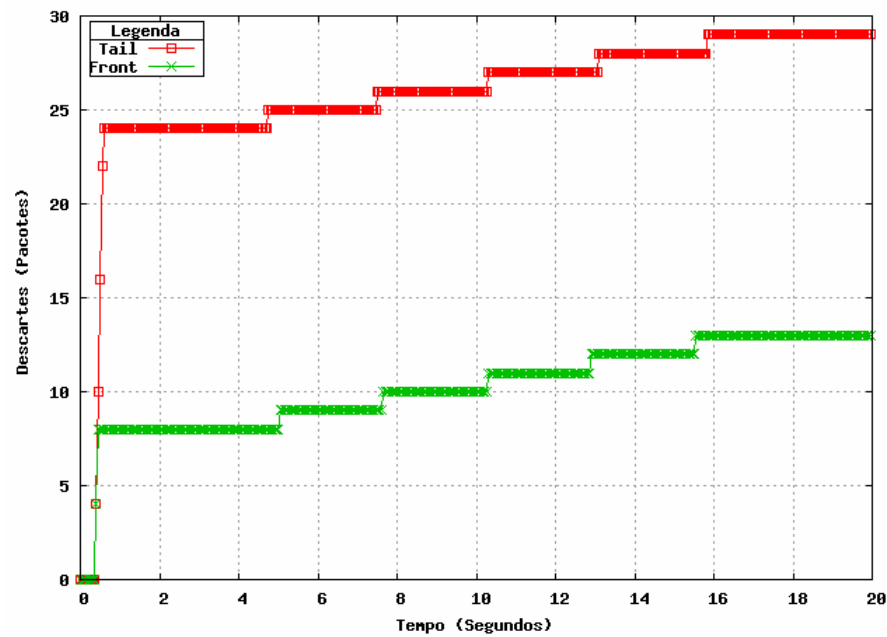


Figura 5.4 – Pacotes descartados na fila, DropTail Vs DropFront

O funcionamento inicial de uma janela de congestionamento no TCP consiste em aumentar o seu tamanho exponencialmente em cada *round trip time* (RTT) até detectar a perda de um pacote, permitindo nesta fase estimar a largura de banda disponível. Nesse instante, metade do valor da janela é armazenado numa variável designada por *ssthresh*. O tamanho da janela é inicializado com esse valor, começando a crescer linearmente a partir desse ponto. Durante o crescimento linear, se voltar a ser detectada uma perda, a janela é novamente inicializada com

metade do seu valor e continua o crescimento linear. O capítulo 6 explora as funcionalidades do controlo de congestionamento do protocolo TCP.

No caso do DropFront, a mensagem do primeiro pacote descartado chega mais rapidamente ao emissor. Essa notificação faz com que o período inicial finalize mais rapidamente e que menos pacotes sejam descartados nesse período (pois o emissor inicializa o tamanho da sua janela). No caso do DropTail, a mensagem do primeiro pacote descartado demora mais tempo a chegar ao emissor. A demora dessa notificação faz com que o período inicial se prolongue e que mais pacotes sejam descartados, como se pode verificar na figura 5.4.

UDP

Para o caso do UDP é utilizada a mesma topologia de rede, alterando os agentes TCP por UDP e a aplicação FTP por uma fonte de tráfego que gera pacotes de tamanho fixo igual a 500 *bytes*, com intervalos de tempo entre os pacotes seguindo uma distribuição exponencial e com uma taxa de transmissão de 4 *Mbit/s*. Os resultados obtidos são apresentados na tabela 5.2.

DropTail	Largura de banda utilizada (%)	97.52
	Tamanho médio da Fila (em Pacotes)	18
	Largura Banda (<i>bit/s</i>)	975200
	Descarte (%)	[74,68; 74,94]
	Atraso Max. (s)	0,11
	Atraso Médio (s)	0,10
DropFront	Largura de banda utilizada (%)	97.52
	Tamanho médio da Fila (em Pacotes)	18
	Largura Banda (<i>bit/s</i>)	975200
	Descarte (%)	[74,59; 74,85]
	Atraso Max. (s)	0,07
	Atraso Médio (s)	0,05

Tabela 5.2 – Resultados obtidos com UDP, DropTail Vs DropFront

Verifica-se que os 4 primeiros parâmetros analisados são iguais, ou seja, independentes da técnica de descarte escolhida; os últimos dois parâmetros são diferentes para cada uma das técnicas. Descartar o primeiro pacote da fila (técnica DropFront), ou seja o próximo pacote a transmitir, permite reduzir o atraso máximo e médio comparando com a técnica DropTail, como é possível verificar pelos resultados obtidos na tabela 5.2. No caso do DropFront, o descarte do pacote vai influenciar o cálculo do atraso máximo e mínimo, diminuindo esses

valores; por outro lado, no caso do DropTail, o descarte do pacote que acaba de chegar não influencia os cálculos dos atrasos na fila de espera.

5.3.2 RED - Parâmetros da fila de espera

Esta secção tem por objecto estudar o efeito da variação dos limiares da técnica de descarte RED. O cenário escolhido é igual ao utilizado na secção anterior com o protocolo de transporte TCP, alterando apenas a técnica de descarte presente na fila de espera para RED.

Variando o limiar inferior e superior dos parâmetros da fila de espera RED, obtém-se o gráfico apresentado na figura 5.5 para a ocupação média na fila.

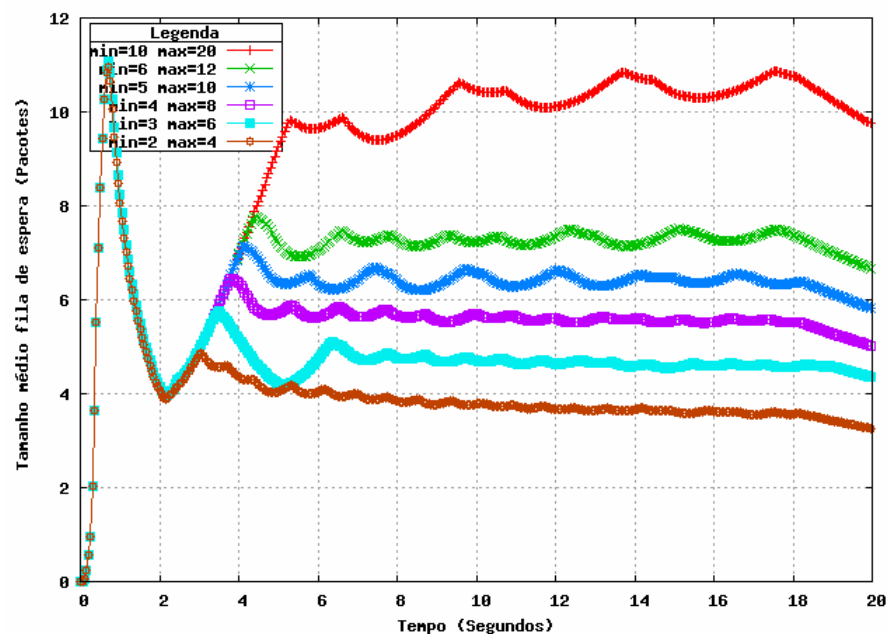


Figura 5.5 – Tamanho médio variando os parâmetros do RED (a legenda segue a ordem do gráfico)

Os dois segundos iniciais do gráfico sofrem a influência da fase de controlo de congestionamento do protocolo de transporte TCP, em que o protocolo estima a largura de banda disponível. A partir dos dois segundos visualiza-se o efeito da técnica de descarte RED no controlo do tamanho médio da fila de espera, estabilizando o tamanho médio da fila de espera entre o valor do limiar inferior (min) e do limiar superior (max) configurado.

5.3.3 DropTail Vs RED - Condição para o descarte de pacotes

Nesta secção apresenta-se a comparação entre a técnica de descarte RED e a técnica de descarte DropTail na presença do protocolo de transporte TCP. O cenário escolhido para esta secção é apresentado na figura 5.6.

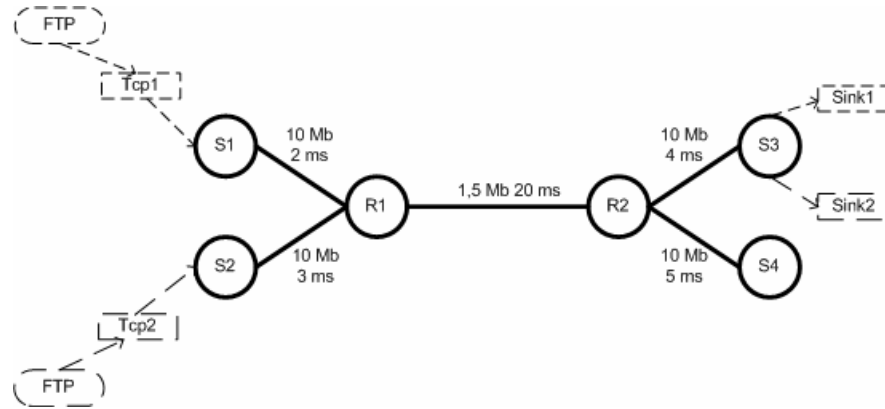


Figura 5.6 – Cenário da Simulação, DropTail Vs RED

Os agentes de origem TCP da figura, TCP1 e TCP2, são do tipo Reno. Os agentes de destino, Sink1 e Sink2, são agentes do tipo TCP/Sink. O tamanho máximo das janelas de transmissão de cada agente de origem é de 15 pacotes. O agente TCP1 inicia a transmissão de dados com FTP para o Sink1 no instante zero. Três segundos depois é iniciada a transmissão de dados com FTP do agente TCP2 para o Sink2. A simulação corre durante 10 segundos. No caso da técnica de descarte RED, configura-se o limiar mínimo da fila de espera igual a 5 pacotes e o limiar máximo igual a 15 pacotes. O tamanho máximo da fila de espera é igual a 25 pacotes.

A figura 5.7 apresenta os valores médios e instantâneos de ocupação da fila de espera para as técnicas de descarte RED e DropTail.

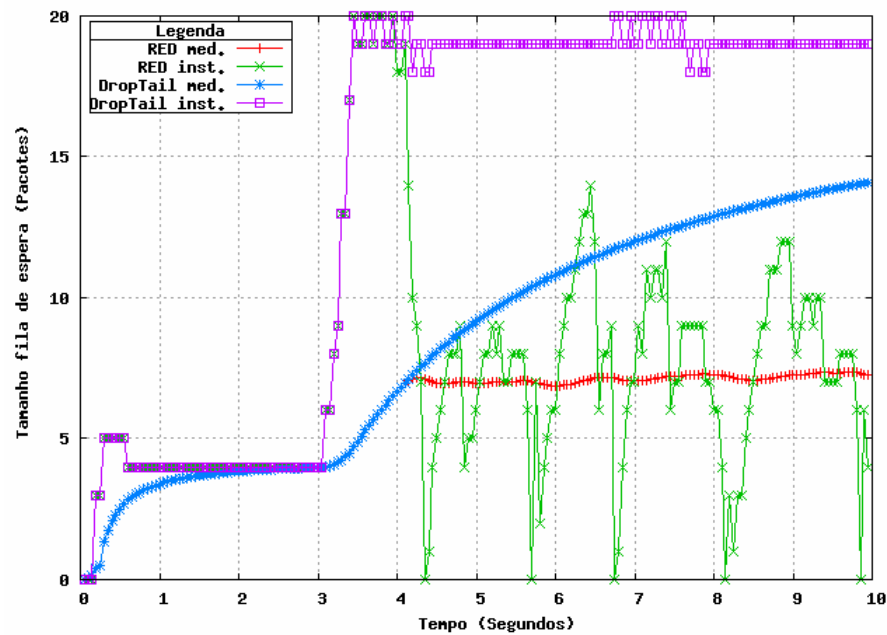


Figura 5.7 – Ocupação da fila de espera, DropTail Vs RED

Os 4 segundos iniciais dos dois gráficos são iguais, ou seja, no processo de *Slow Start* inicial, o tipo de fila de espera não interfere na ocupação. No período no qual apenas o fluxo 1 está activo, o valor médio e instantâneo da ocupação são iguais a 4 pacotes. No intervalo dos 3 aos 4 segundos, o valor instantâneo aumenta para valores entre 19 e 20 pacotes e o valor médio cresce linearmente. A partir dos 4 segundos os gráficos diferem: no caso do RED, o valor médio estabiliza no valor de 7 pacotes e o valor instantâneo oscila, enquanto que no DropTail o valor instantâneo estabiliza nos 19 pacotes e o valor médio cresce. Em ambas as situações a capacidade da fila de espera não é excedida.

De modo a melhor entender o que cada fluxo contribui para a ocupação da fila de espera a partir dos 4 segundos, é analisada a informação relativa à janela de congestionamento de cada fluxo, obtendo o gráfico da figura 5.8.

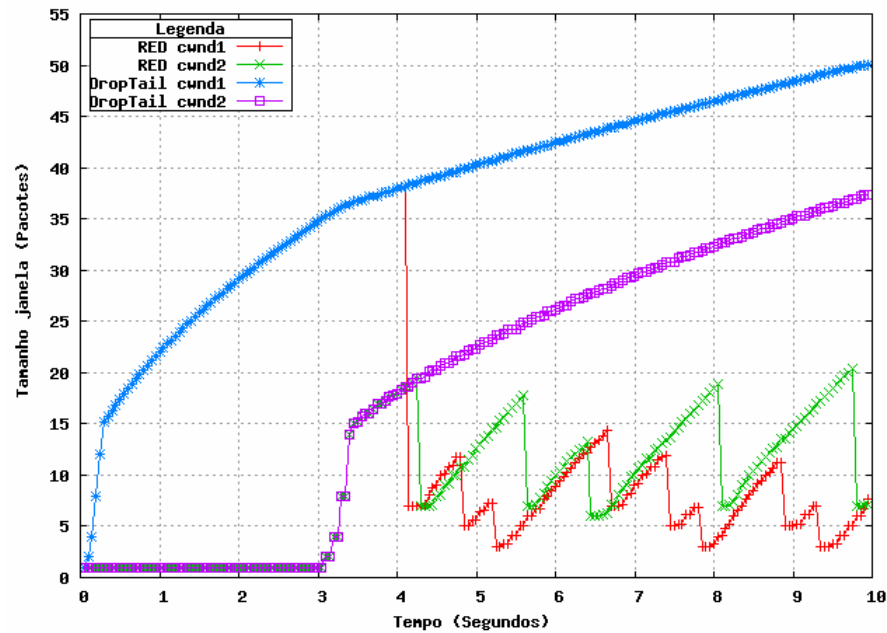


Figura 5.8 – Janela de Congestionamento, DropTail Vs RED.

O valor efectivo da janela de cada agente é igual ao valor mínimo entre o tamanho máximo da janela de transmissão, designado por $Awnd$ (*Advertised Window*) e o tamanho da janela de congestionamento, designada por $Cwnd$ (*Congestion Window*). Como já foi referido neste exemplo, a fila de espera não descarta pacotes devido à limitação da sua capacidade. Verifica-se no gráfico 5.8 que, no caso do DropTail, as janelas de congestionamento crescem exponencialmente até atingir o valor de $Awnd$, igual a 15 pacotes. Atingindo esse ponto, a janela efectiva é constante e igual a 15 pacotes, o que explica o valor estável do tamanho instantâneo da fila de espera e o crescimento do valor médio do tamanho da fila de espera. No caso do RED, a partir dos 3.5 segundos, o valor médio da fila de espera ultrapassa o limiar inferior, e por isso, os pacotes que chegam são descartados com uma certa probabilidade. Pelo gráfico da figura 5.8 verifica-se que é no instante 4 que o primeiro pacote é descartado (fluxo 1). A técnica de descarte RED faz com que a partir desse ponto, o valor médio da fila de espera estabilize, mas o descartar de pacotes provoca uma oscilação na janela de congestionamento dos fluxos, o que explica a oscilação do valor instantâneo da fila de espera.

O cenário apresentado anteriormente não representa uma situação de congestionamento, e nesse cenário em particular a utilização da técnica DropTail é mais eficaz que a técnica de descarte RED. O cenário é alterado de modo a poder comparar essas duas técnicas numa situação de congestionamento: nesse caso a largura de banda da ligação é diminuída (de 1,5

Mbps para 1 Mbps) e o número máximo de pacotes de dados na rede é aumentado, passando de 30 para 50 pacotes. A figura 5.9 apresenta o novo cenário.



Figura 5.9 – Novo Cenário da Simulação, DropTail Vs RED

A nova simulação apenas será efectuada com um único fluxo, de forma a eliminar interferências que possam existir entre diferentes fluxos. O agente TCP origem é do tipo Reno com o tamanho máximo da sua janela igual a 50 pacotes; o agente TCP destino é do tipo TCP/Sink. Este cenário simula a transferência de 2 Mbytes de dados com a aplicação FTP. No caso da técnica de descarte RED configura-se o limiar mínimo da fila de espera igual a 4 pacotes e o limiar máximo igual a 8 pacotes. O tamanho máximo da fila de espera é igual a 20 pacotes.

Na tabela 5.3 são apresentados os resultados obtidos.

	Pacotes Enviados	Pacotes Retransmitidos	ACK Recebidos	LB (Kbits/s)	Fim de dados na fila (s)
DropTail	2054	53	2025	912.490	18.46
RED	2063	62	2026	912.940	18.46

Tabela 5.3 – Resultados obtidos para o novo cenário, DropTail Vs RED

Analisando os dados obtidos, verifica-se que as técnicas apenas diferem no número de pacotes retransmitidos, e por consequência, no número de pacotes enviados. Embora o RED retransmita mais pacotes, em ambos os casos, o último pacote abandona a fila no mesmo instante. Os dados apresentados na tabela 5.3 não são conclusivos. Sendo assim, são analisados graficamente a evolução temporal da ocupação da fila de espera, do valor de RTT e da janela de congestionamento para ambas as técnicas, respectivamente nas figuras 5.10, 5.11 e 5.12.

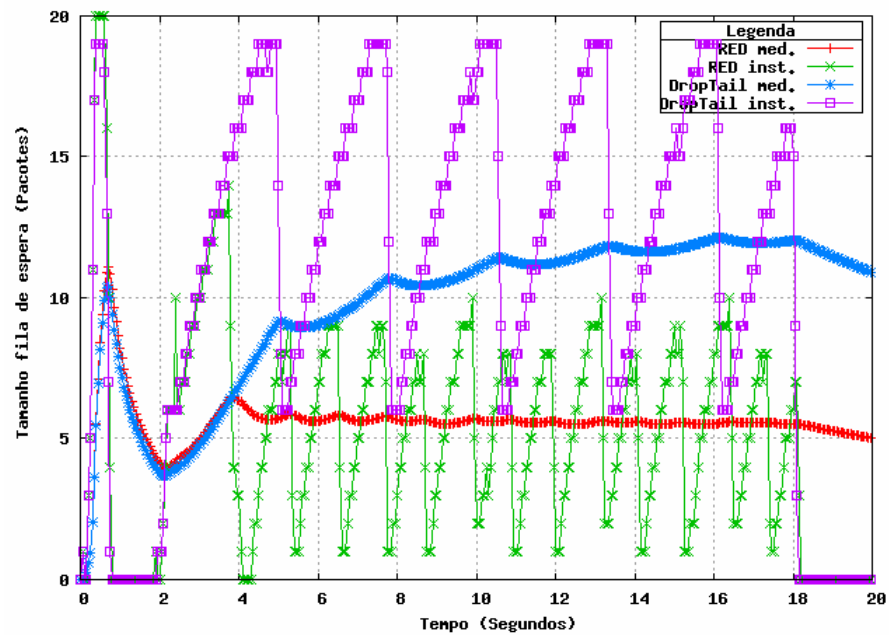


Figura 5.10 – Ocupação instantânea e média da fila de espera no novo cenário DropTail Vs RED

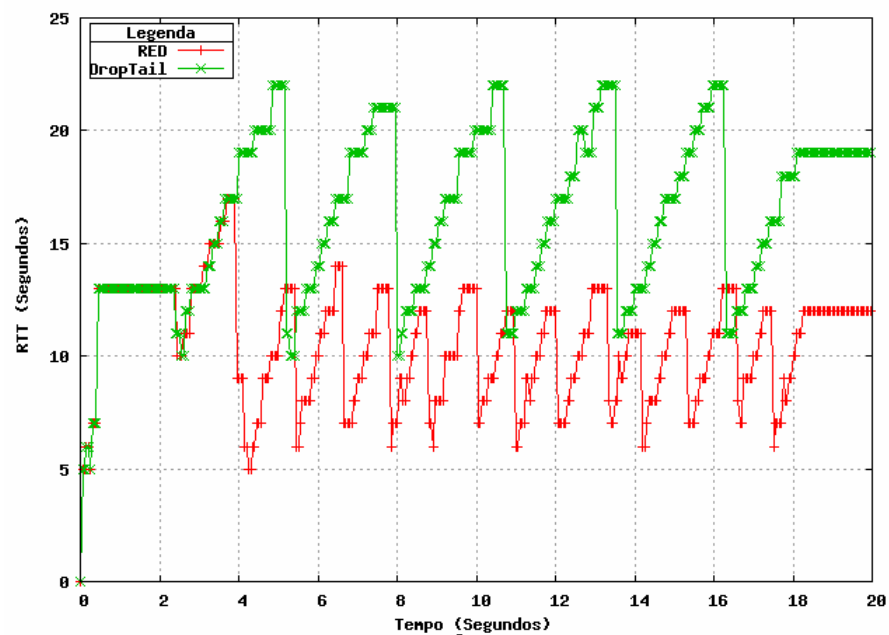


Figura 5.11 – Evolução do RTT no novo cenário DropTail Vs RED

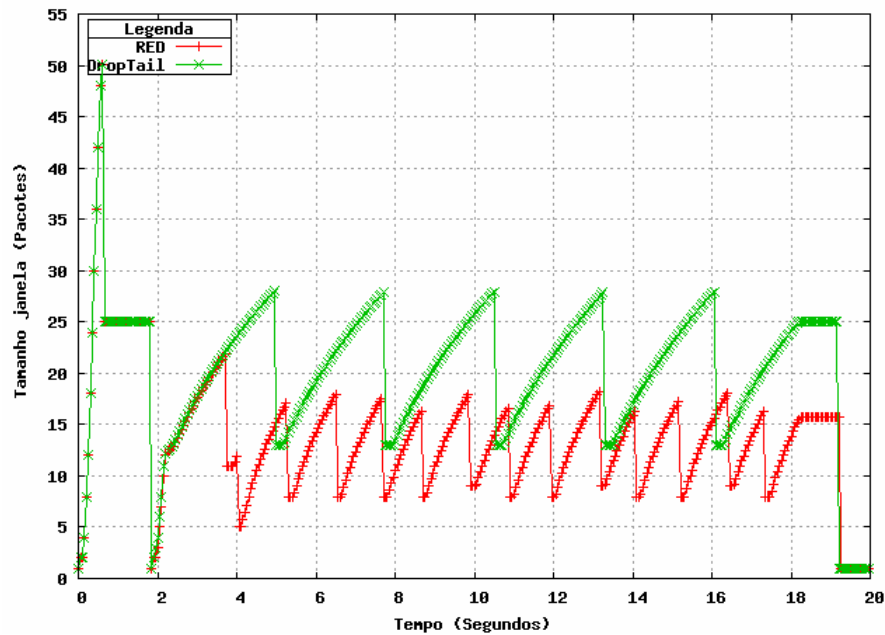


Figura 5.12 – Janelas de Congestionamento no novo cenário DropTail Vs RED

No gráfico da ocupação da fila de espera (figura 5.10) verifica-se a eficiência da técnica RED ao controlar o valor médio da fila entre os valores de limiar predefinidos ($\min = 4$ pacotes e $\max = 8$ pacotes), fazendo com que o valor instantâneo esteja abaixo de metade da capacidade da fila passados os instantes iniciais. No caso do DropTail não existe qualquer tipo de controlo: a capacidade da fila é sucessivamente excedida e no mínimo existem sempre 6 pacotes na fila, provocando maiores variações no valor RTT do que no RED, como pode ser constatado na figura 5.11. Na janela de congestionamento do RED (figura 5.12), o número de vezes que a janela é quebrada deve-se aos pacotes que são descartados com uma certa probabilidade durante a simulação, enquanto que no DropTail as quebras coincidem com os instantes em que a capacidade da fila de espera é excedida.

Verifica-se neste cenário que a técnica de descarte RED permite aperfeiçoar a gestão do controlo da janela de congestionamento, controlando a ocupação média nas filas de espera e por consequência os atrasos dos pacotes na rede e o nível de congestionamento da mesma.

5.3.4 DropTail Vs RED - Sincronização global

Para o estudo do problema da sincronização global das fontes na presença do protocolo TCP acrescentam-se mais 3 fluxos com as mesmas características do fluxo já existente no cenário da figura 5.8. Analisando a evolução temporal das janelas de congestionamento de cada fluxo,

obtiveram-se os gráficos apresentados nas figuras 5.13 e 5.14, para a técnica de descarte DropTail e RED respectivamente.

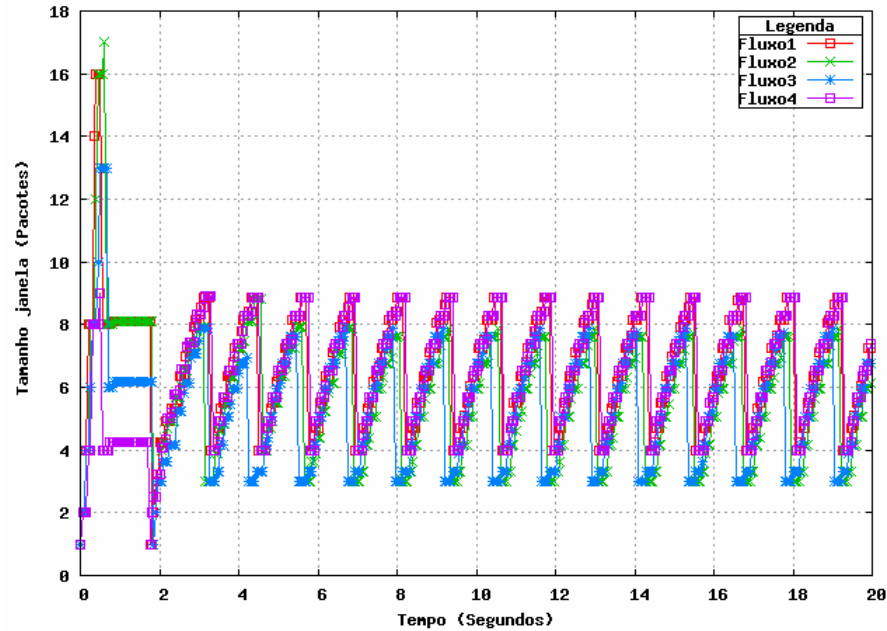


Figura 5.13 – Janelas de Congestionamento dos 4 fluxos, DropTail

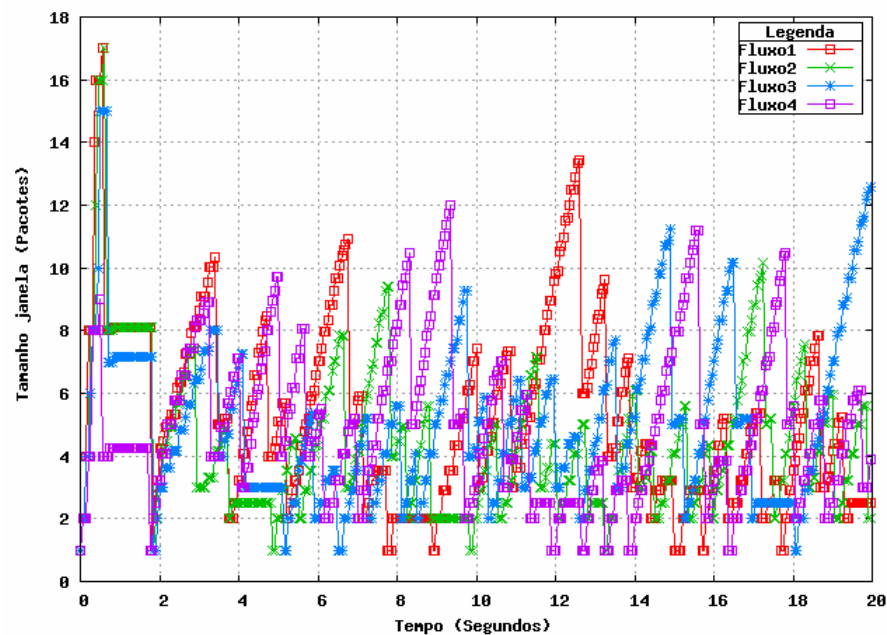


Figura 5.14 – Janelas de Congestionamento dos 4 fluxos, RED

Neste cenário simples com apenas 4 fluxos é possível observar um dos grandes objectivos da técnica de descarte RED, a eliminação da sincronização global. Na figura 5.13, para o caso da técnica de descarte DropTail, verifica-se que as 4 fontes estão síncronas, diminuindo e aumentando as suas janelas de congestionamento em simultâneo. Esse efeito de sincronização

desaparece quando utilizada a técnica de descarte RED: na figura 5.14 confere-se que as 4 fontes já não estão síncronas. Para verificar se a sincronização global tem efeito na largura de banda das ligações, apresentam-se as larguras de banda da ligação R1-R2 para os casos da técnica DropTail e RED, respectivamente nas figuras 5.15 e 5.16.

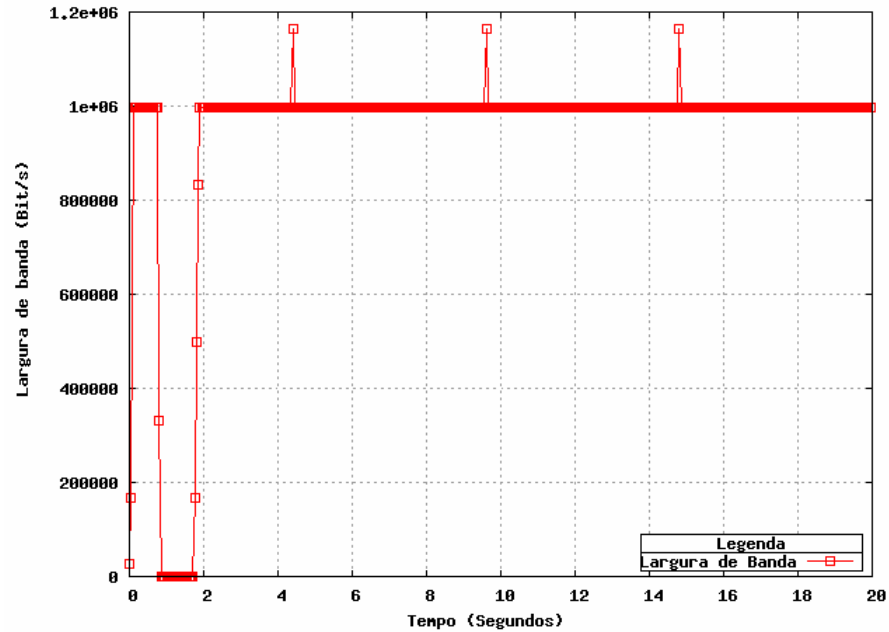


Figura 5.15 – Ocupação da largura de banda da ligação com 4 fluxos, DropTail

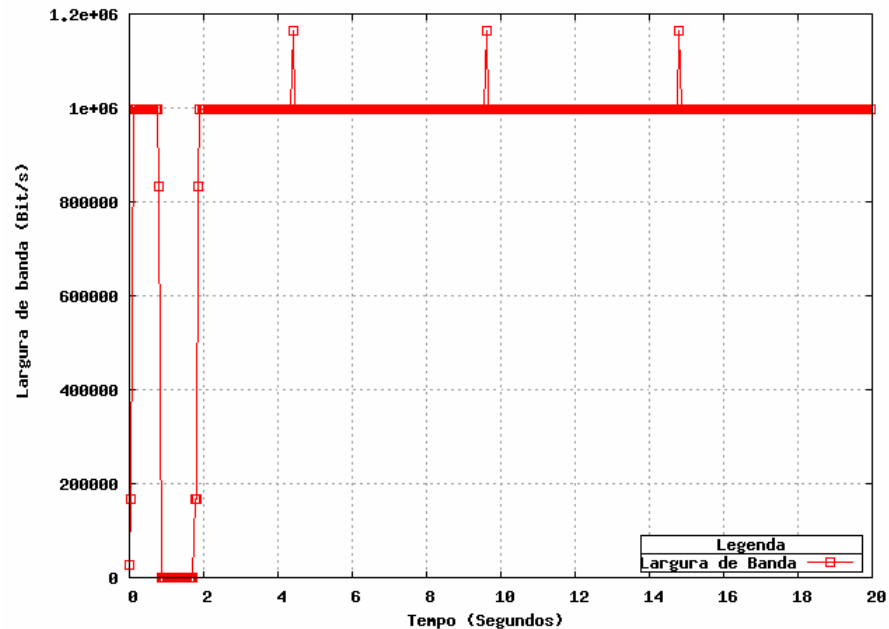


Figura 5.16 – Ocupação da largura de banda da ligação com 4 fluxos, RED

Verifica-se que, ao nível da largura de banda utilizada, não é perceptível o efeito da sincronização global como esperado. A figura 5.15 deveria apresentar quedas na largura de banda sempre que a janela de congestionamento dos quatro fluxos da figura 5.13 quebrasse. De forma a perceber o que está a acontecer neste cenário em particular, analisou-se a ocupação da fila de espera ao longo do tempo apresentada na figura 5.17.

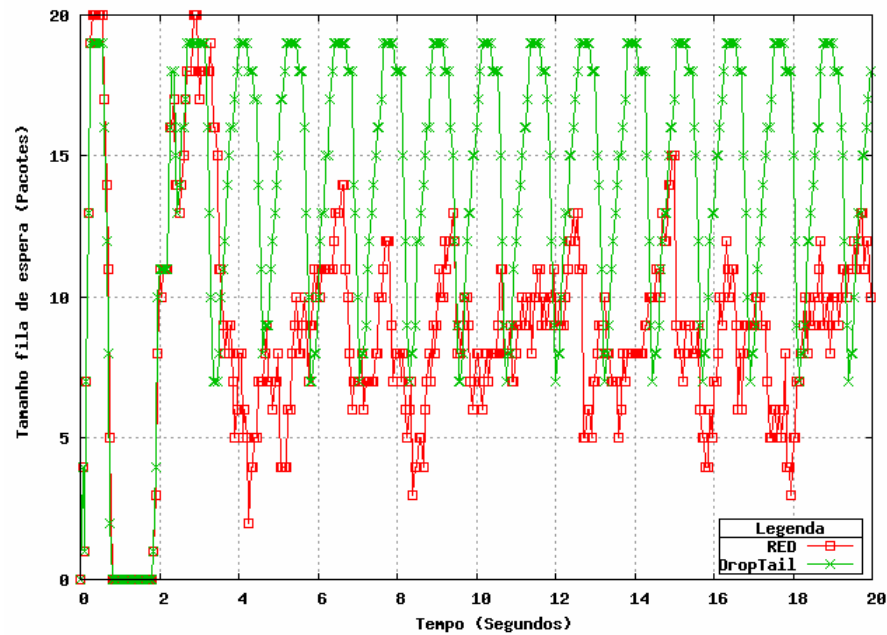


Figura 5.17 – Ocupação da fila de espera na presença dos 4 fluxos

Em ambos os casos, e excluindo os dois primeiros segundos que correspondem à fase inicial do TCP, existem sempre pacotes na fila de espera para serem transmitidos, ou seja, a ligação nunca deixa de receber pacotes. Deste modo, a ocupação da largura de banda da ligação é sempre constante, daí a não visualização do efeito da sincronização global. Nos instantes de reinicialização dos parâmetros do TCP, se não existissem pacotes à espera de transmissão, o efeito de diminuição drástica da largura de banda seria visível caso fosse utilizada a técnica de descarte DropTail.

5.4 Conclusões

Neste capítulo foram estudadas, através de simulação, as técnicas de descarte DropTail, DropFront e RED.

Primeiro, compararam-se as técnicas de descarte DropTail e DropFront, que diferem apenas na escolha do pacote a descartar da fila de espera. No caso da comparação efectuada com o protocolo de transporte TCP constatou-se que, descartar o primeiro pacote da fila de espera com a técnica DropFront, faz com que a notificação de que o pacote foi descartado chegue mais rapidamente ao emissor. No caso do protocolo de transporte UDP, o facto de descartar o primeiro pacote da fila de espera, ou seja, o que está há mais tempo na fila, faz com que o atraso médio e máximo dos pacotes na fila de espera sejam inferiores.

Da análise da técnica de descarte RED, verificou-se o efeito da variação do limiar superior e inferior parameterizáveis da fila de espera. A técnica de descarte RED controla o tamanho médio da fila de espera, estabilizando esse valor entre o valor do limiar inferior e do limiar superior configurados.

Por último, compararam-se as técnicas de descarte DropTail e RED para o caso do protocolo de transporte TCP. A técnica de descarte RED surge como uma mais valia face à técnica de descarte DropTail em cenários de congestionamento, onde os recursos das filas de espera têm que ser eficientemente geridos. A técnica de descarte RED controla o tamanho médio da fila de espera, notificando os emissores para diminuírem as suas janelas quando necessário, evitando assim cenários de congestionamento da rede.

Na presença de mais que um fluxo e com a técnica de descarte DropTail verifica-se o efeito da sincronização global. As janelas de congestionamento de todos os fluxos estão síncronas, crescendo e diminuindo ao mesmo tempo, o que significa um desaproveitamento dos recursos existentes. A técnica de descarte RED elimina o efeito de sincronização global, porque os fluxos são notificados aleatoriamente para diminuírem o tamanho das suas janelas, através do descarte de um dos seus pacotes.

6. Controlo de congestionamento

O congestionamento da rede surge quando ocorre uma saturação dos recursos das ligações da mesma. O controlo de congestionamento tem como objectivo reagir e combater essa situação; executa-se através de um conjunto de mecanismos implementados nos nós de uma rede e nos extremos (emissores e receptores).

O controlo de congestionamento pode ser efectuado em malha aberta ou em malha fechada. Em malha aberta, os emissores não recebem sinais de congestionamento da rede, mas a admissão de novas ligações é controlada com reguladores de tráfego do tipo *Leaky-Bucket*, ou reservando recursos de acordo com o pedido do emissor. Em malha fechada, os emissores adaptam-se dinamicamente ao estado da rede, recebendo sinais de congestionamento da rede que podem ser implícitos ou explícitos. Os sinais explícitos são, por exemplo, as perdas de pacotes ou as variações do valor de RTT (*Round-Trip Time*), medidos através dos pacotes de confirmação (ACKs); os sinais implícitos são, por exemplo, a utilização do *bit* ECN (*Explicit Congestion Notification*) do cabeçalho dos pacotes. Ambos estes sinais são utilizados no protocolo de transporte TCP (*Transmission Control Protocol*). O valor de RTT é o tempo que o pacote demora a percorrer o caminho do emissor para o receptor, e o tempo que o pacote de confirmação demora a regressar ao emissor [14].

No caso do controlo de congestionamento em malha aberta existem algumas dificuldades na configuração *a priori* dos mecanismos de controlo, quando ainda não se conhecem as características da rede. Devido a estas dificuldades, muitas vezes as redes são dominadas por algoritmos de controlo de congestionamento em malha fechada.

Actualmente os mecanismos do TCP constituem um exemplo de controlo de congestionamento na Internet. O TCP é o protocolo predominante para transporte de tráfego em redes de computadores, incluindo a Internet. Tem como função garantir a entrega dos pacotes entre o emissor e o receptor, tendo em conta e reagindo (através dos seus mecanismos de controlo de congestionamento) ao estado de congestionamento da rede.

O objectivo deste capítulo é analisar ao longo do tempo a evolução do protocolo TCP, estudando alguns dos seus mecanismos, começando por ordem cronológica pela primeira versão especificada no RFC793, passando pela Tahoe, de seguida pela Reno e finalmente a

Vegas. Através de um cenário de simulação são observados os mecanismos de controlo de congestionamento de cada uma das versões, bem como as principais diferenças entre elas.

Este capítulo está organizado da seguinte forma. Na secção 6.1 são apresentados os conceitos básicos do TCP. A evolução dos algoritmos para o cálculo do valor de *timeout* é analisada na secção 6.2. A secção 6.3 apresenta os mecanismos de controlo de congestionamento. As simulações efectuadas e os resultados obtidos são descritos na secção 6.4. As conclusões deste capítulo são apresentadas na secção 6.5.

6.1 Introdução

No TCP, o primeiro acontecimento antes da transferência de dados entre o emissor e o receptor é o estabelecimento da ligação. Esse processo é efectuado em três passos e é designado por *three-way handshake*. A figura 6.1 e a tabela 6.1 exemplificam passo-a-passo o estabelecimento de uma ligação TCP. O pacote SYN é utilizado para iniciar ou finalizar a transmissão de dados; contém activa uma *flag* do seu cabeçalho, *flag SYN*, que indica o tipo de pacote. O pacote ACK é utilizado pelo receptor sempre que recebe um pacote, de modo a notificar o emissor dessa recepção, indicando qual o próximo pacote que o receptor espera receber; contém activa uma *flag* do seu cabeçalho, *flag ACK*.

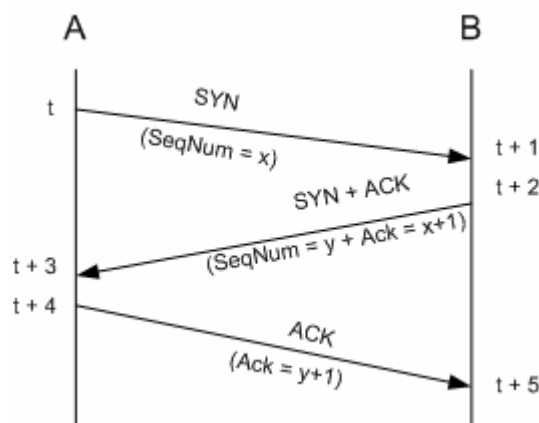


Figura 6.1 – Estabelecer uma ligação TCP

Instante de Tempo	Evento
t	A envia um pacote SYNchronize para B
t + 1	B recebe o pacote SYN de A
t + 2	B envia o seu pacote SYN para A com o ACKnowledge do SYN recebido
t + 3	A recebe o pacote SYN de B e o ACK do SYN enviado
t + 4	A envia um pacote de ACK do SYN recebido
t + 5	B recebe o ACK do SYN enviado e a ligação é estabelecida

Tabela 6.1 – Passos do estabelecimento de uma ligação TCP

Após o estabelecimento da ligação é iniciada a transferência de dados entre o emissor e o receptor; nesta fase podem surgir problemas de congestionamento da rede. Para libertar a ligação, o emissor envia um pacote com a *flag* FIN activa, indicando ao receptor que ele não tem mais dados para enviar. A recepção do pacote FIN é confirmada e a ligação é desactivada. O exemplo apresentado considera uma comunicação unidireccional; no caso de uma comunicação *Full Duplex* o procedimento para ligar e desligar a comunicação é efectuado também no sentido contrário.

A figura 6.2 exemplifica a troca de pacotes entre o emissor e o receptor. Os pacotes enviados pelo emissor são numerados sequencialmente. A cada pacote recebido o receptor envia uma confirmação, pacote com a *flag* ACK activa, na qual indica através do número sequencial qual o próximo pacote que espera receber. No exemplo da figura 6.2, o receptor ao receber o pacote número 2 envia ao emissor um pacote de confirmação, com a indicação que aguarda receber o pacote número 3. Quando o pacote número 3 é perdido, ao receber o pacote número 4, o receptor confirma a recepção do pacote e reenvia ao emissor a indicação que continua a aguardar o pacote número 3. Esse pacote de confirmação é designado por confirmação duplicada.

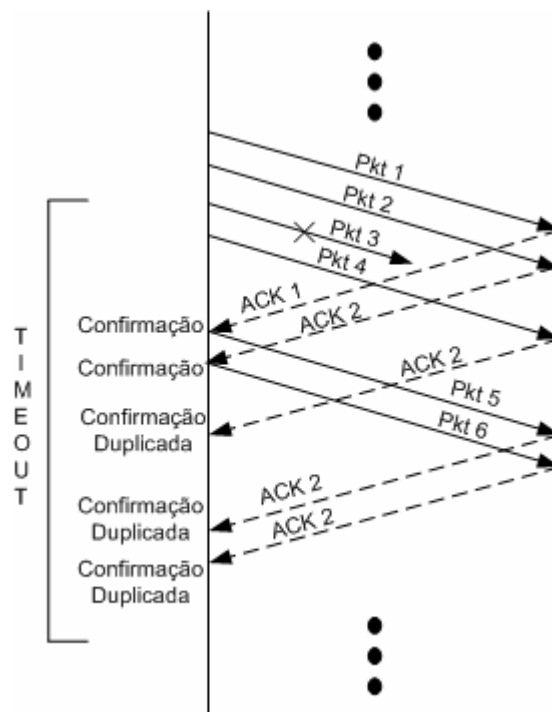


Figura 6.2 – Transferência de dados numa ligação TCP

O emissor TCP deverá possuir mecanismos para limitar o congestionamento da rede. Se existirem perdas de pacotes, o TCP necessita de diminuir o tráfego enviado para a rede;

associados a estes mecanismos de congestionamento existem mecanismos de retransmissão dos pacotes perdidos. Para estudar e compreender os mecanismos de controlo de congestionamento é necessário determinar o instante de tempo em que se deve considerar a existência de congestionamento da rede, e retransmitir os pacotes perdidos. Este instante de tempo é designado por *timeout*. A secção seguinte apresenta os algoritmos utilizados para determinar este tempo.

6.2 Relógio de retransmissão - *timeout*

O TCP tem a função de garantir a entrega dos pacotes entre o emissor e o receptor. O emissor, após enviar um pacote, espera um determinado tempo pela recepção da confirmação desse pacote. Se, ao fim de um determinado intervalo de tempo, o emissor não recebe essa confirmação, então ele retransmite os pacotes não confirmados. Esse intervalo de tempo é designado por *timeout* e é calculado em função do RTT .

Os algoritmos de cálculo do valor de *timeout* evoluíram ao longo do tempo. O primeiro algoritmo, especificado no RFC793, calcula o *timeout* em função do valor médio de RTT . Esse algoritmo tem limitações, nomeadamente ele não diferencia um pacote enviado pela primeira vez de um retransmitido, o que provoca erros no cálculo do *timeout*. Para colmatar essa limitação surge um novo algoritmo designado por algoritmo de Karn. Por fim, surge o algoritmo de Jacobson, que ao contrário dos algoritmos anteriores, calcula o valor de *timeout* tendo em conta, não só o valor médio de RTT , como também a variação desse valor. Os 3 algoritmos são apresentados nas secções seguintes.

6.2.1 Algoritmo Original

Este algoritmo é especificado no RFC793 [9], em que o valor de *timeout* é calculado em função do valor médio de RTT . Cada vez que o TCP envia um pacote, esse instante de tempo é registado. Quando o TCP recebe a confirmação desse pacote, ele calcula o valor dessa amostra $ARTT$ (amostra do RTT). O valor de $ARTT$ é o tempo que a confirmação da recepção desse pacote demora a chegar. O TCP actualiza o valor de RTT segundo a seguinte expressão

$$RTT = \alpha RTT + (1 - \alpha) ARTT \quad (6.1)$$

onde ao valor anterior de RTT é adicionada uma fracção do valor da amostra $ARTT$.

O parâmetro α determina qual o peso do valor RTT anterior e é tipicamente igual a $7/8$. O valor de *timeout* é dado por

$$Timeout = 2 \times RTT \quad (6.2)$$

6.2.2 Algoritmo de Karn

O mecanismo descrito anteriormente tem um problema quando existe retransmissão de pacotes. Como a confirmação contém informação do último pacote recebido com sucesso, o emissor não consegue distinguir se o ACK recebido se refere ao pacote inicial ou ao pacote retransmitido. Esse dado é importante para calcular correctamente o valor de $ARTT$ porque, se esse valor for calculado tendo em conta o pacote errado, esse cálculo pode ser subdimensionado ou sobredimensionado. No exemplo (a) da figura 6.3, o valor de $ARTT$ é determinado tendo em conta o instante de partida do pacote retransmitido e o instante de chegada do pacote inicial. Neste caso, o valor de $ARTT$ é subdimensionado. No exemplo (b), o valor de $ARTT$ é determinado tendo em conta o instante de partida do pacote inicial e o instante de chegada do pacote retransmitido. Neste caso, o valor de $ARTT$ é sobredimensionado.

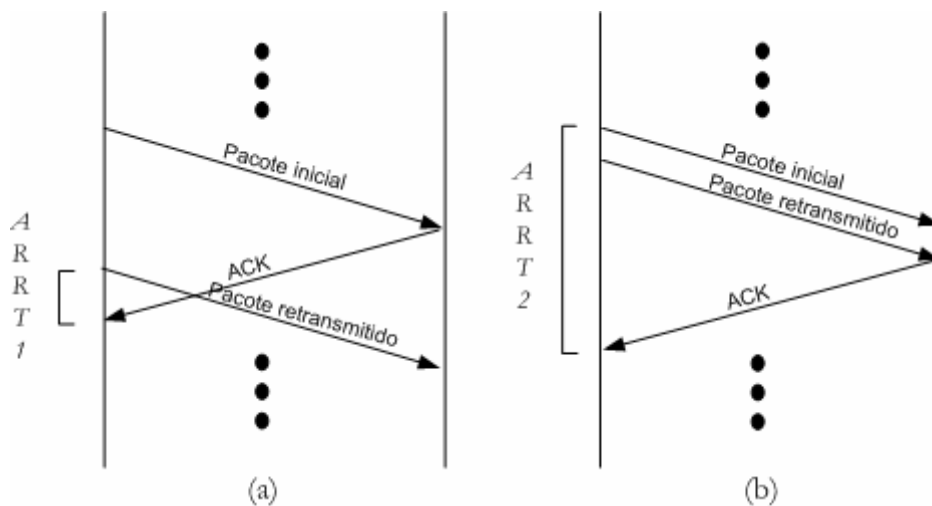


Figura 6.3 – Valor de $ARTT$ calculado de forma errada

A solução proposta por Karn e Partridge consiste em ter em conta apenas os valores de pacotes que não representam retransmissão: sempre que um pacote é retransmitido não é registado o valor de $ARTT$. Nestas situações, o valor de *timeout* é actualizado utilizando um mecanismo designado por *Exponential backoff*. Este mecanismo consiste em duplicar o valor do *timeout* anterior cada vez que o emissor retransmite. A duplicação do *timeout* neste mecanismo

tem em conta o facto de, se existe uma retransmissão, isso significa que existe congestionamento da rede, e os pacotes necessitam de mais tempo para efectuar o mesmo percurso, sendo o valor estimado de *timeout* duplicado.

6.2.3 Algoritmo de Jacobson

Este algoritmo foi introduzido quando a Internet sofria de grandes problemas de congestionamento. O facto de se considerar que houve um *timeout* está relacionado com a situação de congestionamento. Se se considerar a existência de *timeout* antes do instante devido, um pacote será retransmitido desnecessariamente, o que só irá agravar o congestionamento.

O problema com o algoritmo original é que o cálculo do *timeout* é efectuado tendo em conta o valor médio das amostras RTT , não dando importância à variação dessas amostras. Se a variação dessas amostras for pequena, então o valor actualizado de RTT é de confiança e igual ao valor de *timeout*, não sendo necessário multiplicá-lo por dois. Por outro lado, se a variação for grande, o valor de *timeout* não pode ser calculado apenas tendo em conta o valor médio das amostras.

No algoritmo de Jacobson, a variação das amostras RTT , $RTTVAR$, é calculada tendo em conta o valor de $RTTVAR$ anterior, o valor da amostra RTT e o valor de RTT anterior, através da expressão

$$RTTVAR = \alpha RTTVAR + (1 - \alpha) |RTT - RTT| \quad (6.3)$$

onde α é igual ao do algoritmo original. O valor de *timeout* é dado por

$$Timeout = RTT + 4 \times RTTVAR \quad (6.4)$$

Segundo a expressão, quando a variação é pequena, o intervalo de tempo *timeout* é próximo do valor de RTT .

6.3 Mecanismos de Controlo de Congestionamento

O objectivo principal dos mecanismos de controlo de congestionamento é prevenir e/ou resolver problemas de congestionamento na rede. A primeira especificação do TCP é apresentada no RFC793 de 1981 [9], onde é descrita a implementação de um mecanismo de

controlo de fluxo baseado numa janela, em que o tamanho dessa janela define a quantidade máxima de pacotes que pode existir na rede entre o emissor e o receptor. Jacobson e Karels desenvolveram em 1988 o TCP Tahoe [10] [11] que possui dois mecanismos de controlo de congestionamento, *Slow Start* e *Congestion Avoidance*, e um novo mecanismo de retransmissão *Fast Retransmit*. O TCP Reno, criado por Jacobson, surge em 1990 [12] e acrescenta à versão anterior um novo mecanismo designado por *Fast Recovery*. Em 1995 é desenvolvido por Brakmo e Peterson o TCP Vegas [13]. Esta versão do TCP altera os mecanismos *Congestion Avoidance*, *Slow Start* e *Fast Retransmit* presentes nas versões anteriores, permitindo com essa alteração utilizar mais eficientemente a largura de banda disponível na ligação.

As sub-secções seguintes apresentam os diversos mecanismos presentes no TCP. Na sub-secção 6.3.1 é apresentado o princípio de funcionamento do controlo de fluxo do TCP baseado numa janela. Os mecanismos introduzidos na versão TCP Tahoe, *Slow Start*, *Congestion Avoidance* e *Fast Retransmit* são apresentados nas sub-secções 6.3.2, 6.3.3 e 6.3.4, respectivamente. A sub-secção 6.3.5 apresenta o mecanismo introduzido pela versão TCP Reno, designado por *Fast Recovery*. Por fim a sub-secção 6.3.6 apresenta as alterações efectuadas pela versão TCP Vegas em alguns dos mecanismos existentes nas versões anteriores.

6.3.1 *Sliding Window*

A primeira especificação do TCP foi efectuada no RFC793 em 1981, onde é descrita a implementação de um mecanismo de controlo de fluxo baseado numa janela, designada por *Advertised Window (Awnd)*. Este mecanismo de controlo de fluxo é designado por *Sliding Window* e é apresentado na figura 6.4.

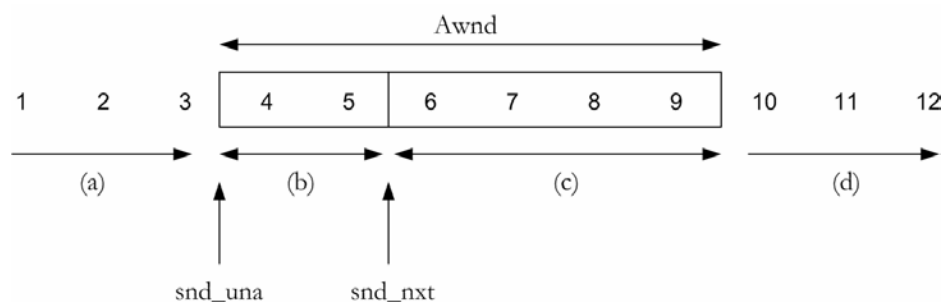


Figura 6.4 – *Sliding Window*

O receptor impõe um tamanho máximo da janela de pacotes que pode receber, *Awnd*, por possuir um espaço de memória limitado para colocar os pacotes recebidos. O emissor não

pode enviar um número de *bytes* superior ao tamanho da janela sem receber confirmação do receptor dos pacotes que recebeu. Na figura 6.4 o tamanho da janela ($Awnd$) é de 6 *bytes* e a janela movimenta-se da esquerda para a direita. Snd_una é o número sequencial do último *byte* não confirmado e snd_nxt é o número sequencial do próximo *byte* a enviar. Na figura, os 3 primeiros *bytes* enviados (a) já foram confirmados pelo receptor, os *bytes* 4 e 5 (b) foram enviados mas o emissor ainda não recebeu as confirmações respectivas, os restantes *bytes* da janela (c) podem ser enviados pelo emissor, e os últimos três *bytes* (d) não podem ser enviados enquanto a janela não deslizar. A relação entre as diversas variáveis da janela $Awnd$ é apresentada na expressão

$$snd_nxt - snd_una + 1 \leq Awnd \quad (6.5)$$

A actualização da janela (expressão 6.6) é feita com base na capacidade de processamento do receptor e pelo tamanho do espaço de memória reservado para os pacotes recebidos, designado por $MaxRcvBuffer$. Os *bytes* colocados na memória do receptor são lidos (retirados) da memória por uma aplicação que processa essa informação. Se a aplicação que utiliza o TCP lê e retira os *bytes* da memória $MaxRcvBuffer$ à mesma taxa que os pacotes são lá colocados, então $Awnd = MaxRcvBuffer$. Se a aplicação for mais lenta, o valor de $Awnd$ é decrementado de modo a informar o emissor do espaço livre ainda disponível. Numa situação limite, em que o receptor não consegue processar mais informação, o valor de $Awnd$ é colocado a zero fazendo com que o emissor não envie mais pacotes até indicação em contrário.

$$Awnd = MaxRcvBuffer - (Ultimo_Byte_recebido - Ultimo_Byte_Lido) \quad (6.6)$$

A janela desliza para a direita cada vez que recebe a confirmação do receptor. Uma confirmação do receptor contém a *flag* ACK activa, a identificação do pacote que recebeu (ou do próximo a receber) e o tamanho da janela $Awnd$ no campo *Window* do cabeçalho. Mesmo que não exista qualquer alteração nesses campos, o receptor reenvia sempre essa informação quando recebe um pacote.

No caso do emissor não ter permissão para enviar mais pacotes, como o receptor só envia pacotes quando os recebe, quando a aplicação do receptor lê pacotes, é necessário existir um mecanismo que informe o emissor que a sua janela já não é zero. Para o efeito, o emissor possui um cronómetro, designado por *Persistence Timer* utilizado para resolver esta situação: ele é inicializado quando o emissor recebe um ACK com $Awnd$ igual a zero; ao fim de um tempo

predefinido pelo cronómetro, o emissor envia um pacote de um *byte* cujo único objectivo é aguardar que o receptor, em resposta a esse pacote, o informe da alteração do valor de *Awnd*. Deste modo, o TCP controla o fluxo de pacotes e avisa o emissor da altura em que este pode continuar a enviar pacotes.

Quando o emissor detecta que se perdeu um pacote, porque o tempo esperado para receber uma resposta do receptor expirou, ele activa o *timeout* e reenvia esse pacote. Um pacote perdido devido ao congestionamento da rede provoca uma reacção do emissor de reenvio desse pacote. Este mecanismo de controlo de fluxo garante que o receptor não recebe mais informação para além da que pode suportar. No entanto, se não existirem outros mecanismos de reacção ao congestionamento da rede, a rede terá um aumento no seu estado de congestionamento. Sendo assim, surge a necessidade de construir mecanismos que permitam aos emissores TCP controlar os seus fluxos de modo a não agravar o congestionamento já existente.

6.3.2 *Slow Start*

Este mecanismo, designado por *Slow Start*, é utilizado para controlar a taxa de envio de pacotes para a rede de modo a que o emissor não a congestionue. Neste mecanismo são introduzidas duas novas variáveis para a ligação: a *Congestion Window* (*Cwnd*) e uma variável que representa o limiar da fase *Slow Start* (*ssthresh*).

O valor da janela do emissor é definido pelo valor mínimo entre *Awnd* e *Cwnd*. O valor de *Awnd* faz com que o emissor não envie mais pacotes do que o receptor possa receber; o valor de *Cwnd* é utilizado para que o emissor não envie mais pacotes do que a rede possa suportar. O tamanho da janela *Cwnd* é adaptado à carga da rede, diminuindo o seu valor quando a rede começa a ficar congestionada e aumentando-o quando a rede está mais livre. Na prática, o emissor sabe que a rede está congestionada quando detecta que alguns dos seus pacotes foram descartados.

No mecanismo *Slow Start*, quando uma ligação é estabelecida, o tamanho da janela *Cwnd* é igual à unidade. Esse valor é incrementado de um pacote por cada confirmação recebida. Nesta fase, o tamanho da janela cresce exponencialmente, como pode ser observado na figura 6.5. Durante este período o TCP está a tentar aprender qual a quantidade de largura de banda que está disponível na ligação.

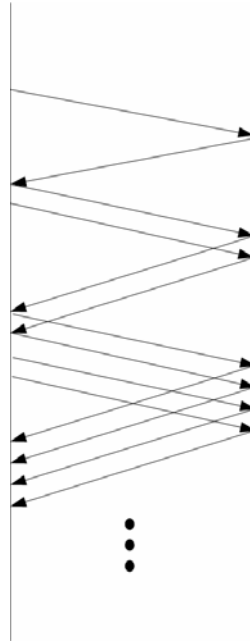


Figura 6.5 – Evolução dos pacotes durante a fase *Slow Start*

O valor inicial do limiar da fase *Slow Start* ($ssthresh$) é igual a um valor elevado, fazendo com que a janela cresça exponencialmente até detectar uma perda. Nesse instante, metade do valor da janela é armazenada na variável $ssthresh$ e o mecanismo de *Slow Start* é reiniciado até atingir o valor do limiar calculado. A partir desse limiar a janela aumenta de uma forma linear (na próxima secção é apresentada esta fase, designada de *Congestion Avoidance*). Deste modo, o *Slow Start* inicial é utilizado para calcular o valor de $ssthresh$.

Dado que o aumento da janela neste mecanismo é efectuado de uma forma exponencial, o nome *Slow Start* é algo estranho. Essa designação tem sentido quando comparada com a implementação do TCP inicial. Originalmente, o TCP enviava (se tivesse pacotes para enviar) de uma só rajada o tamanho da sua janela $Awnd$, o que poderia provocar congestionamento se os nós não tivessem capacidade para processar essa quantidade de pacotes. O novo mecanismo impede a ocorrência dessas rajadas de pacotes.

6.3.3 *Congestion Avoidance*

Quando o tamanho da janela atinge o valor do limiar $ssthresh$, por cada ACK recebido, o tamanho de $Cwnd$ é incrementado de $1/(\text{tamanho da janela})$, o que equivale a incrementar a janela de um pacote por RTT, como pode ser observado na figura 6.6. Este mecanismo incrementa linearmente o tamanho da janela até à ocorrência de uma perda.

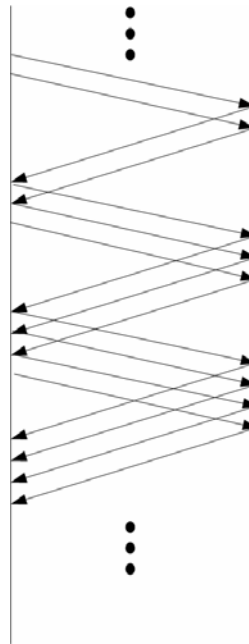


Figura 6.6 – Evolução dos pacotes durante a fase *Congestion Avoidance*

Na prática, os mecanismos de *Slow Start* e *Congestion Avoidance* completam-se: por cada ACK recebido o emissor segue o seguinte algoritmo apresentado na figura 6.7, dependendo dos valores da janela de congestionamento e do limiar *ssthresh*.

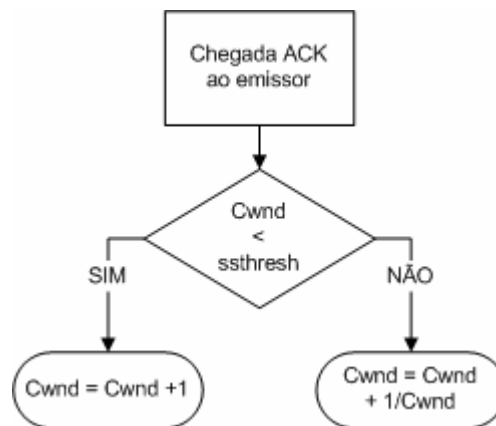


Figura 6.7 – Fluxograma *Slow Start* + *Congestion Avoidance*

Quando chega uma confirmação ao emissor, este verifica os valores de *Cwnd* e *ssthresh*, actualizando o valor da sua janela de congestionamento segundo o fluxograma da figura 6.7: caso $Cwnd < ssthresh$, ou seja, caso se encontre na fase *Slow Start*, aumenta a janela no valor de 1 segmento; caso $Cwnd > ssthresh$, ou seja, caso se encontre na fase *Congestion Avoidance*, incrementa a janela no valor de $1/\text{tamanho da janela}$. Quando uma perda é detectada pelo emissor em qualquer uma das fases, *Slow Start* ou *Congestion Avoidance*, o emissor decreta *timeout*, o valor de *Cwnd* é colocado a 1 e a variável *ssthresh* assume metade do valor de *Cwnd* antes da

perda. O emissor inicia a transmissão na fase *Slow Start* e o processo de aumento da janela prossegue seguindo o algoritmo do fluxograma acima descrito.

6.3.4 *Fast Retransmit*

No início da implementação do TCP, este apenas detectava as perdas de pacotes por *timeout*. Se a confirmação do pacote enviado chega ao emissor antes de o *timeout* expirar, então o emissor sabe que a rede não está excessivamente congestionada. Se este tempo expirar e o emissor não tiver recebido a respectiva confirmação, então é decretado *timeout* e o emissor volta a enviar o pacote. As perdas detectadas por *timeout* introduzem tempos mortos no envio dos pacotes, porque o emissor tem que aguardar um tempo igual ao valor de *timeout* para identificar que um determinado pacote se perdeu.

O mecanismo de *Fast Retransmit* foi introduzido de modo a reduzir esses tempos mortos. Se por algum motivo o receptor recebe um pacote que não é o que ele espera, ou seja, tem um número de sequência superior ao esperado, volta a enviar o pacote de ACK com a mesma indicação que o que enviou anteriormente, informando que espera receber um pacote com sequência anterior. O emissor, por sua vez, interpreta o ACK repetido como uma possível chegada não ordenada dos pacotes ao receptor e continua a enviar os seus pacotes normalmente. Após a recepção de três vezes do mesmo ACK repetido, o emissor apercebe-se que o pacote realmente se deve ter perdido e reenvia-o, não esperando que o *timeout* expire para o fazer. Esta situação está exemplificada na figura 6.8. No exemplo, o pacote número 3 é perdido. O receptor, ao receber o pacote número 4, indica na confirmação (enviada ao emissor), que continua à espera de receber o pacote número 3 (1º ACK duplicado). O receptor, ao receber os pacotes números 5 e 6, envia o segundo e terceiro ACK duplicado, respectivamente. O emissor, ao receber o terceiro ACK duplicado, reenvia o pacote número 3.

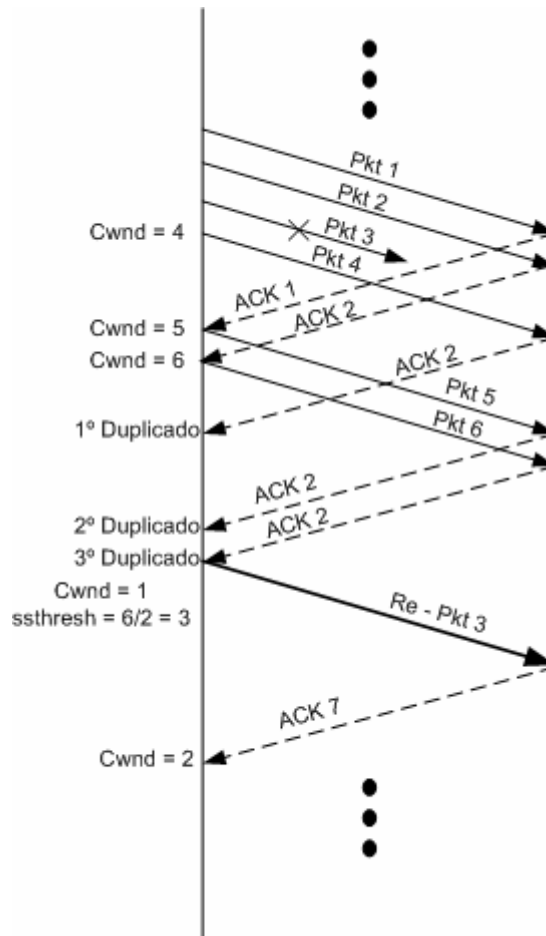


Figura 6.8 – ACK duplicados

As perdas detectadas por ACK duplicados não ocorrem quando o tamanho da janela $Cwnd$ é pequeno, visto que neste caso o período de *timeout* expiraria antes que um emissor recebesse os três ACK duplicados.

6.3.5 Fast Recovery

Este novo mecanismo vem completar o mecanismo de *Fast Retransmit*. Quando as perdas são detectadas por *timeout*, o valor de $Cwnd$ é colocado a 1 e é iniciada a fase *Slow Start*. Quando as perdas são detectadas por três ACK repetidos (*Fast Retransmit*), o mecanismo de *Fast Recovery* é accionado. Neste caso, a fase *Slow Start* não acontece, o valor de $Cwnd$ é colocado igual ao valor de $ssthresh$ (metade do valor de $Cwnd$ quando iniciaram as confirmações duplicadas), e continua a fase *Congestion Avoidance*. A fase *Slow Start* apenas ocorre no início da ligação e quando ocorrem perdas detectadas por *timeout*.

A recepção de ACK duplicados indicam que, mesmo tendo sido perdidos alguns pacotes, o receptor está a receber pacotes, ou seja, a rede não está muito congestionada. A ocorrência de

um *timeout* indica que o receptor não está a receber pacotes, ou seja, a rede está muito congestionada. A fase *Slow Start* só se justifica no início da ligação, quando é necessário calcular a largura de banda disponível, e em situações de muito congestionamento, fazendo com que os emissores TCP não contribuam para esse congestionamento. A figura 6.9 apresenta um exemplo com o mecanismo *Fast Recovery* activo.

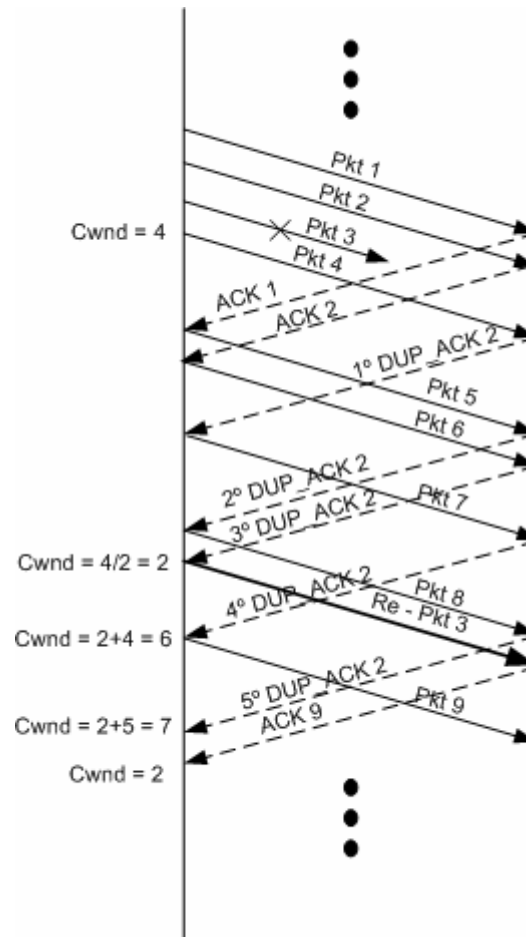


Figura 6.9 – Fast Recovery

No exemplo da figura 6.9, o pacote 3 é perdido. Após a recepção do terceiro ACK repetido, designado na figura por 3º DUP_ACK 2, o emissor entra no modo *Fast Retransmit*, e efectua as seguintes operações: retransmite o pacote 3, referente ao ACK repetido, coloca *ssthresh* igual a metade da janela *Cwnd* actual, e *Cwnd* igual a *ssthresh*. Os ACK repetidos seguintes são utilizados para ritmar a transmissão de pacotes, em que o tamanho da janela = $\min (Awnd, Cwnd + ndup)$, e em que *ndup* é o número de ACK duplicados. O valor de *ndup* é igual a zero se o número de pacotes repetidos for menor que 3, e é incrementado aquando da recepção de cada ACK repetido. Quando o emissor recebe um ACK de um novo pacote, chamado *Recovery ACK*, sai do modo *Fast Recovery*, coloca *ndup* = 0 e continua na fase *Congestion Avoidance*.

6.3.6 Alterações efectuadas pelo TCP Vegas

Em 1994 foi desenvolvida uma nova versão do TCP, TCP Vegas, que altera o mecanismo de *Congestion Avoidance* presente nas versões anteriores. O objectivo principal é obter um melhor aproveitamento da largura de banda disponível na ligação, utilizando um esquema mais sofisticado para estimar a largura de banda disponível. Nas versões anteriores, esse cálculo era baseado apenas na detecção de pacotes perdidos, indicando o congestionamento da ligação [19]. O TCP Vegas também efectua alterações no modo *Slow Start* inicial e no mecanismo de retransmissão.

Congestion Avoidance

O TCP Vegas utiliza a diferença entre a taxa de transmissão esperada e a actual para calcular a largura de banda disponível na ligação. Idealmente, quando a ligação não está congestionada, o valor da taxa de transmissão esperada deve ser próxima da actual [17].

O emissor efectua em cada RTT as seguintes operações

$$Taxa_Esperada = \frac{Cwnd}{BaseRTT} \quad (6.7)$$

$$Taxa_Actual = \frac{Cwnd}{RTT} \quad (6.8)$$

$$Diff = (Taxa_Esperada - Taxa_Actual)BaseRTT \quad (6.9)$$

BaseRTT é igual ao mínimo RTT calculado. O valor *Diff* indica a diferença entre as duas taxas, a esperada e a actual. Baseado no valor de *Diff*, o emissor actualiza a sua janela de congestionamento (*Cwnd*) segundo a expressão 6.10. O valor *Diff* é sempre positivo ou igual a zero, pois o valor *BaseRTT* é o valor de RTT mínimo obtido. Os limiares α e β representam o número mínimo e o número máximo de pacotes que o TCP Vegas coloca nas filas de espera, respectivamente. O valor *BaseRTT* é actualizado sempre que ocorre um *timeout*, ou quando apenas existe um pacote transmitido.

$$Cwnd = \begin{cases} Cwnd + 1 & \text{se } Diff < \alpha \\ Cwnd - 1 & \text{se } Diff > \beta \\ Cwnd & \text{se } \alpha < Diff < \beta \end{cases} \quad (6.10)$$

O princípio de funcionamento do TCP Vegas consiste em colocar nas filas de espera da rede pelo menos α pacotes mas nunca mais que β pacotes, tentando deste modo aproveitar a largura de banda disponível sem congestionar a rede.

Comparando o TCP Vegas com o TCP Reno, o segundo está sempre a actualizar o tamanho da sua janela de congestionamento de modo a garantir a utilização total da largura de banda disponível, levando a perdas constantes de pacotes. O TCP Vegas não provoca oscilações ao tamanho da janela de congestionamento, mas sim, converge para um ponto de equilíbrio. Quando o TCP Vegas compete com um TCP Reno na partilha de recursos de uma ligação, o TCP Vegas não recebe a sua parte da largura de banda respectiva devido ao seu mecanismo de *Congestion Avoidance* bastante conservativo. Esta partilha de recursos será analisada na secção 6.4.2.

Slow Start

O mecanismo de *Slow Start* também sofre uma pequena alteração, de modo a evitar as perdas provocadas no *Slow Start* inicial. No arranque não se sabe qual a largura de banda disponível na rede, sendo complicado predefinir o valor de *ssthresh* de modo a que não ocorram perdas nesta fase. Em versões anteriores, o valor inicial de *ssthresh* era elevado e a janela *Cwnd* crescia exponencialmente até detectar perdas. A variável *ssthresh* era então igual a metade do valor da janela de congestionamento quando detectadas as perdas. Para as fases *Slow Start* seguintes, esse problema não surgia porque já era conhecida a largura de banda disponível. Se em vez de se considerar um valor elevado de *ssthresh*, se considerar um valor pequeno, o crescimento exponencial da janela irá parar muito cedo, fazendo com que leve muito tempo a atingir um valor óptimo para *Cwnd* (sendo o seu crescimento linear). Sendo assim, é necessário arranjar um método que evite a existência de perdas no *Slow Start* inicial. No *Slow Start* do TCP Vegas, em vez de a janela de congestionamento crescer exponencialmente todos os RTT, esse crescimento só é efectuado em RTT alternados. Nos RTT em que a *Cwnd* não cresce exponencialmente, o valor da janela de congestionamento é mantido fixo de modo a poder comparar a taxa Actual com a taxa Esperada, calculadas da mesma forma que no mecanismo de *Congestion Avoidance*.

Quando a diferença entre a taxa Esperada e a Actual for inferior ao valor de γ (que por defeito é 1), o TCP Vegas comuta do modo *Slow Start* para o *Congestion Avoidance* [18]. O valor da janela de congestionamento é dado por

$$Cwnd = \begin{cases} Cwnd - \frac{Cwnd}{8} & \text{se } Cwnd \geq 2 \\ 2 & \text{se } Cwnd < 2 \end{cases} \quad (6.11)$$

Mecanismo de retransmissão

No novo mecanismo de retransmissão, o emissor grava o instante de tempo de cada pacote enviado. Quando recebe um ACK, grava também esse instante, e utiliza esse valor com o valor anteriormente guardado do pacote enviado relativo a esse ACK para calcular o RTT . O TCP Vegas utiliza esse valor de RTT mais exacto para decidir a retransmissão nas duas situações seguintes.

- Quando um pacote duplicado chega, o Vegas verifica se a diferença entre o instante da recepção do ACK e o instante do envio do pacote à espera de um ACK é maior que o valor de *timeout*. Se sim, reenvia o pacote sem ter que aguardar por três ACK duplicados;
- Após a retransmissão, quando recebe um primeiro ou um segundo ACK não duplicado, volta a verificar se o intervalo de tempo entre o envio do pacote do próximo ACK esperado e este instante é maior que o valor de *timeout*. Se sim, reenvia o pacote sem ter que aguardar por três ACK duplicados.

6.4 Simulações

O objectivo desta secção é estudar por simulação a evolução dos mecanismos de congestionamento e das diversas versões de TCP.

As simulações efectuadas estão divididas em duas secções: a primeira secção compara entre si as cinco versões de TCP: TCP/RFC793edu, TCP (sem *Fast Retransmit*), TCP/Tahoe, TCP/Reno e TCP/Vegas, analisando o efeito dos mecanismos de congestionamento introduzidos em cada versão no desempenho da rede. A segunda secção compara em mais pormenor a versão de TCP Vegas com a TCP Reno.

O cenário escolhido é apresentado na figura 6.10. A fila de espera à saída do nó R1 é de tamanho igual a 10 pacotes; as restantes filas de espera são de tamanho igual a 50 pacotes. Cada simulação tem a duração de 20 segundos. O tamanho dos pacotes é de 1400 *bytes*. As versões de TCP analisadas são: TCP/RFC793edu, TCP (sem *Fast Retransmit*), TCP/Tahoe, TCP/Reno e TCP/Vegas com $\alpha = 1$, $\beta = 3$ e $\gamma = 1$. Todos os agentes TCP estão

configurados com o tamanho da janela *Window* igual a 35 pacotes, ou seja, no máximo para cada ligação TCP não existirá mais que 35 pacotes de dados simultaneamente na rede.

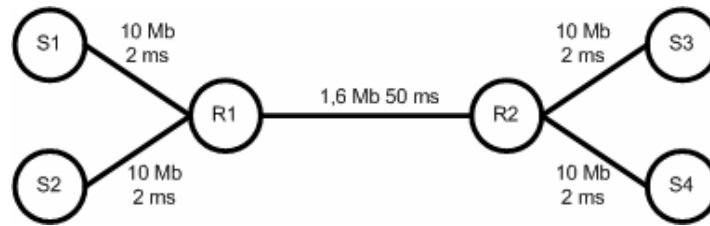


Figura 6.10 – Cenário da Simulação para comparar as versões TCP

As configurações e variáveis de estado presentes no NS para cada um dos agentes TCP estão descritas no anexo B.3.

6.4.1 Comparação entre as versões de TCP

Numa primeira fase de simulações apenas é estabelecido um fluxo entre os nós S1 e S3. Uma aplicação FTP é utilizada para transferir 1Mbytes de dados entre esses dois pontos. Durante as simulações são colecionados os seguintes resultados apresentados na tabela 6.2.

	Pacotes Enviados	Pacotes Retransmitidos	ACK Recebidos	LB (Kbits/s)	Fim de dados na fila (s)
RFC793	259	129	149	97.783	16.96
Tahoe sem <i>FastRTx</i>	750	34	736	850.580	9.96
Tahoe	750	34	736	1128.250	7.50
Reno	742	26	731	1050.583	8.00
Vegas	715	0	715	1333.555	6.00

Tabela 6.2 – Resultados obtidos na comparação das diversas versões de TCP

Para o envio de 1Mbytes de dados com pacotes de 1400 bytes é necessário enviar 715 pacotes. Analisando os pacotes enviados e retransmitidos na versão de TCP especificada no RFC793, verifica-se que aproximadamente metade dos pacotes enviados são pacotes retransmitidos. Sem mecanismos de controlo de congestionamento o TCP apenas consegue utilizar cerca de 6% da largura de banda disponível e não consegue transmitir toda a informação durante os 20 segundos da simulação. As versões seguintes já possuem mecanismos de controlo de congestionamento que permitem enviar toda a informação durante os 20 segundos da simulação. Comparando o TCP Tahoe sem e com o mecanismo *Fast Retransmit* verifica-se que os resultados obtidos de pacotes enviados, retransmitidos e ACK repetidos são iguais. A influência do mecanismo *Fast Retransmit* é sentida na rapidez da transmissão dos dados (coluna 5) e no melhor aproveitamento da largura de banda (coluna 4), dado que não é necessário aguardar pelo tempo de *timeout* para retransmitir. Comparando o TCP Tahoe com o TCP Reno verifica-se que os resultados obtidos não correspondem ao esperado: seria de esperar

que o novo mecanismo (*Fast Recovery*) introduzido pelo TCP Reno permitisse uma transmissão mais rápida dos dados, já que a fase *Slow Start* é eliminada. A análise da evolução temporal das janelas de congestionamento permitirá explicar o porquê de isto estar a acontecer: o envio de pacotes e suas perdas no processo *Slow Start* inicial. Por último o TCP Vegas é a versão que melhor se adaptou ao cenário escolhido, apenas transmitindo os 715 pacotes necessários, em menos tempo que as restantes versões e efectuando um melhor aproveitamento da largura de banda. O estudo da evolução temporal das janelas de congestionamento e da utilização da largura de banda permitirá entender melhor estes valores.

As várias versões TCP são analisadas graficamente através da evolução das janelas de congestionamento, com a excepção da versão RFC793 (que não possui uma janela de congestionamento na sua especificação), permitindo visualizar o efeito da implementação dos novos mecanismos. A figura 6.11 apresenta a evolução temporal da janela de congestionamento (e a janela anunciada *Annnd*) da versão TCP Tahoe sem e com *Fast Retransmit*, da TCP Reno e da TCP Vegas, respectivamente.

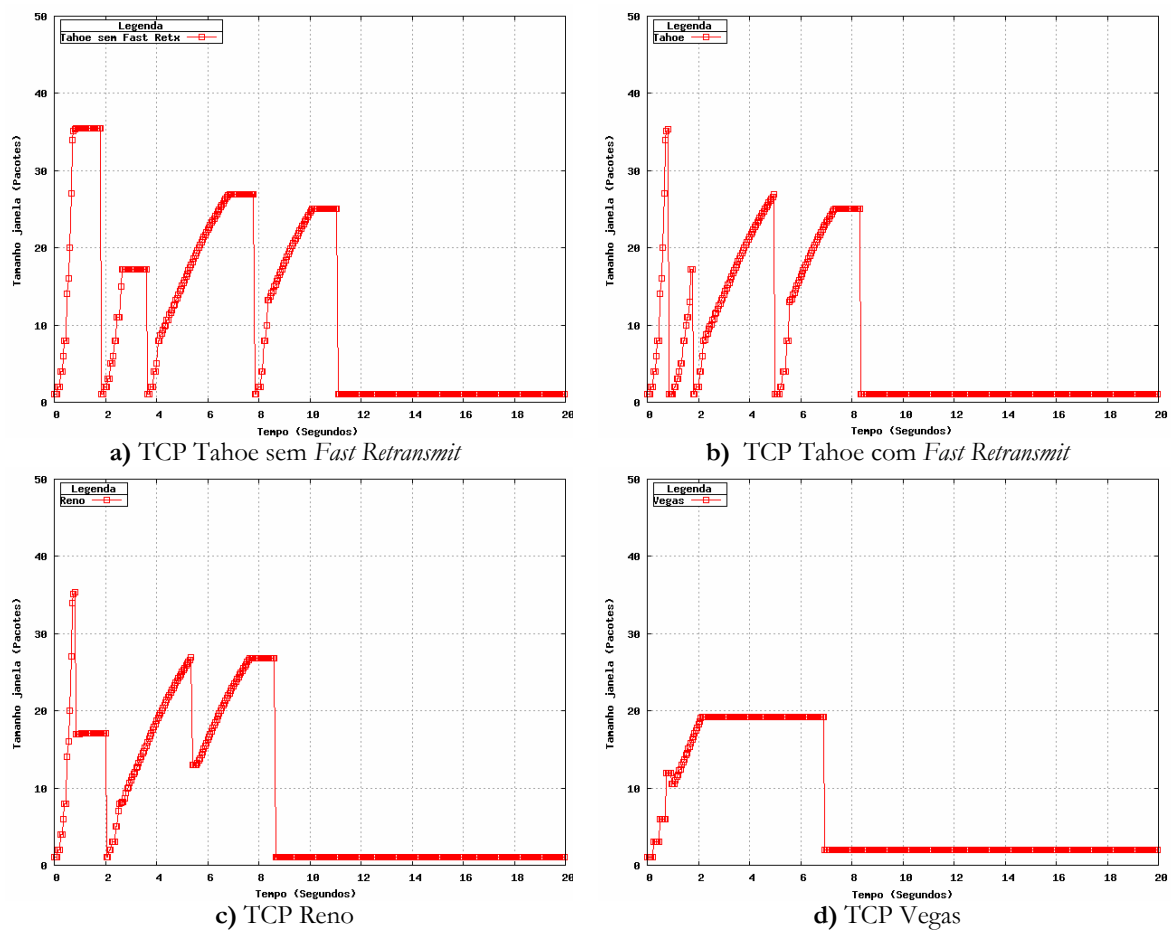


Figura 6.11 – Janela de congestionamento

Analisando as alíneas da figura 6.11, verifica-se que no *Slow Start* inicial a janela cresce exponencialmente em cada RTT até detectar uma perda, de modo a calcular o valor de *ssthresh* inicial. O valor correcto para comutar para a fase seguinte sem perdas pode não ser calculado no primeiro *Slow Start*, mas sim em *Slow Start* seguintes. Analisando os gráficos a) e b) da figura 6.11, o valor *ssthresh* calculado no primeiro *Slow Start* (intervalo de tempo [0,2]s na figura a) e [0,1]s na figura b)) é suficientemente elevado para existir uma perda na fase *Slow Start* seguinte (intervalo de tempo [2,4]s na figura a) e [1,2]s na figura b)), e só na terceira fase *Slow Start* (instante 4 segundos na figura a) e instante 2 segundos na figura b)) é que se comuta para a fase *Congestion Avoidance*. No caso do TCP Vegas apresentado na alínea d) figura 6.11, a fase *Slow Start* inicial cresce exponencialmente em RTT alternados, não necessitando de uma nova fase de *Slow Start* para comutar para a fase seguinte (*Congestion Avoidance*). Quando comuta para a fase *Congestion Avoidance*, o valor da sua janela decresce $cwnd/8$, como pode ser observado na figura d) no instante 1 segundo. Verifica-se que o TCP Vegas consegue estimar a largura de banda disponível mais facilmente e de forma menos agressiva.

Comparando os mecanismos Tahoe sem e com *Fast Retransmit* (alínea a) e b)), quando é detectada uma perda pelo emissor com o mecanismo *Fast Retransmit* activo, não é necessário aguardar um intervalo de tempo igual ao *timeout*. Assim que o emissor recebe três ACK duplicados, ele coloca a janela a 1 e inicia o mecanismo *Slow Start*. Sem o mecanismo *Fast Retransmit* presente, ao detectar uma perda o emissor deixa de transmitir e aguarda um *timeout* para reiniciar a transmissão na fase *Slow Start* com a sua janela igual a 1. Os intervalos de tempo onde o valor de *Cwnd* se mantém constante (na figura a)) representam os tempos de espera da expiração do *timeout* (designados por tempos mortos já que o emissor não transmite dados). O mecanismo *Fast Retransmit* elimina esses tempos mortos como é exemplificado na figura b).

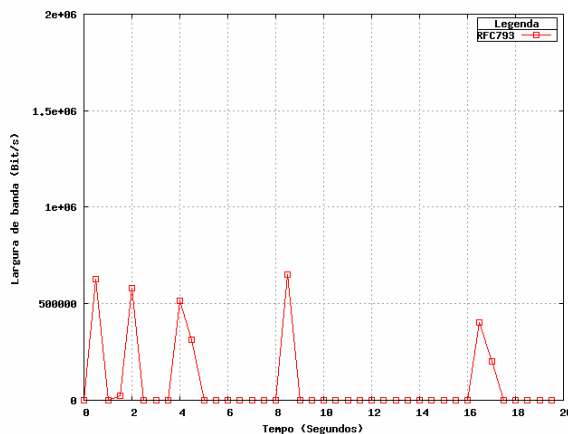
Comparando o gráfico da figura b) com o da figura c), analisando os 2 segundos iniciais (fase *Slow Start* inicial) o *timeout*, que ocorre no caso do TCP Reno (figura c)) durante cerca de 1 segundo, explica o porquê da transmissão com o TCP Reno ter sido (neste exemplo) mais lenta que com o TCP Tahoe (tabela 6.2). No intervalo de tempo [4,6]s, verifica-se o execução do mecanismo *Fast Recovery*. No caso em que se utiliza o TCP Reno (figura c)), a janela *Cwnd* não é inicializada a 1 quando detectada uma perda, mas assume metade do valor anterior e volta a crescer linearmente (fase *Congestion Avoidance*). Na situação em que se utiliza o TCP

Tahoe (figura b)), a fase *Congestion Avoidance* é interrompida ao detectar uma perda e a janela de congestionamento é colocada a 1, crescendo de seguida exponencialmente (fase *Slow Start*).

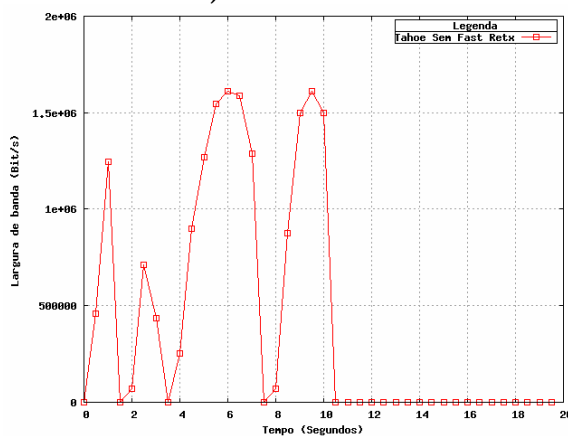
Comparando o TCP Reno com o TCP Vegas, verifica-se que as oscilações constantes do tamanho da janela desaparecem no TCP Vegas, o que significa um menor número de pacotes perdidos. Esse comportamento deve-se à atitude menos agressiva do TCP Vegas na estimação da largura de banda disponível.

Em todos os exemplos, quando ocorre uma perda por expirar o *timeout*, visualizada no gráfico pelo valor constante da janela de congestionamento, a janela de congestionamento fica com o valor 1 e a transmissão é inicializada com a fase *Slow Start*.

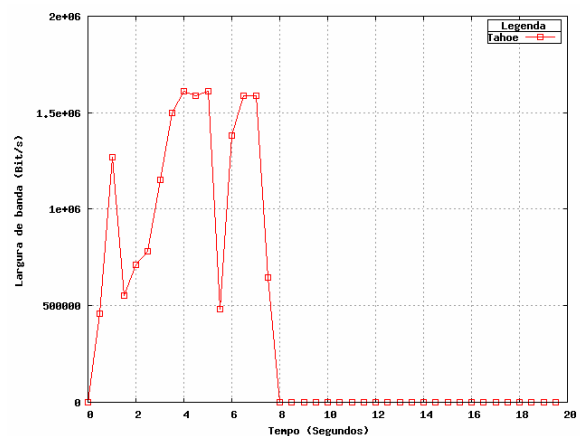
A figura 6.12 apresenta a evolução temporal da largura de banda utilizada na ligação R1-R2, para a versão de TCP do RFC793, TCP Tahoe sem e com *Fast Retransmit*, TCP Reno e TCP Vegas, respectivamente. Este estudo gráfico permite um melhor entendimento das diferenças entre as diversas versões.



a) TCP RFC793



b) TCP Tahoe sem *Fast Retransmit*



c) TCP Tahoe com *Fast Retransmit*

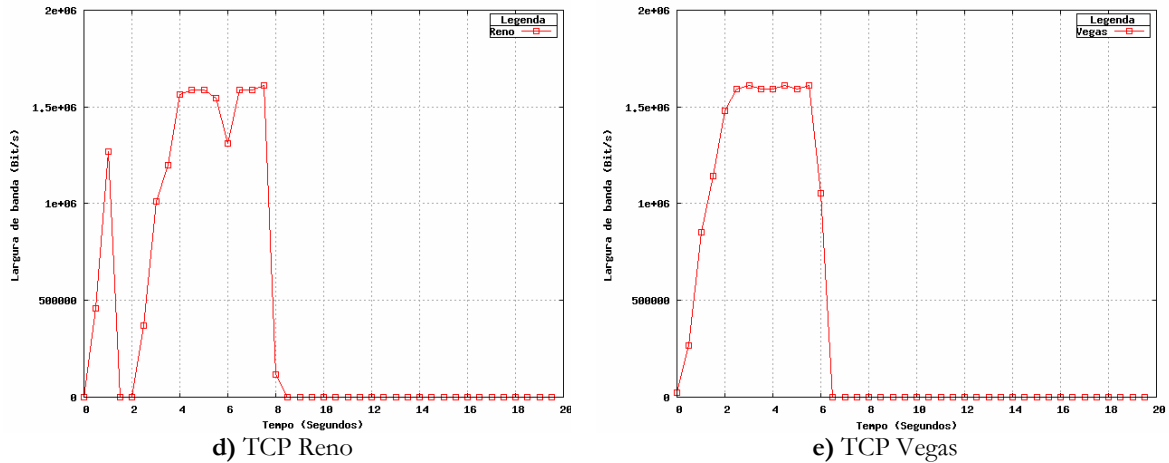


Figura 6.12 – Largura de banda da ligação

Na alínea a) da figura 6.12 verifica-se que a maior parte dos pacotes enviados no caso do TCP RFC793 não chegam ao emissor, ou seja, são descartados na fila de espera. No gráfico os picos correspondem ao tipo de tráfego gerado pelo TCP RFC793 (rajadas de pacotes enviados pelo emissor) que são descartados após a ocupação total da fila de espera. Comparando a figura b) com a c), verifica-se o efeito dos tempos mortos introduzidos pelos *timeout*: Com o mecanismo *Fast Retransmit* inactivo (figura b)) a curva da largura de banda sofre maior número de quedas e mais acentuadas; com o mecanismo *Fast Retransmit* activo (figura c)) a curva da largura de banda nunca atinge o valor nulo, ou seja, existe um melhor aproveitamento da largura de banda. Na alínea d) da figura 6.12, comparando o instante 6 do gráfico com o mesmo instante do gráfico c), visualiza-se o efeito do mecanismo *Fast Recovery* onde, no caso deste mecanismo estar activo, a quebra da ocupação da largura de banda é muito menos acentuada. Quando as perdas são detectadas por *timeout*, a transmissão é sempre inicializada na fase *Slow Start*, fazendo com que a largura de banda utilizada nesse instante seja menor. Por último, a figura e) demonstra o melhor aproveitamento da largura de banda do TCP Vegas.

Para avaliar melhor o desempenho do TCP Vegas em comparação com o TCP Reno são efectuadas simulações mais completas na secção seguinte, com diferentes condições iniciais do sistema.

6.4.2 TCP Reno Vs TCP Vegas

Esta secção tem por objectivo comparar com mais pormenor o TCP Reno e o TCP Vegas. Este estudo endereça a variação de um conjunto de parâmetros e condições iniciais da

simulação, tais como o atraso de propagação, a capacidade da fila de espera e o número de fluxos presentes, de modo a estudar a influência dessas alterações nas duas versões do TCP.

Utilizando o cenário da alínea anterior, comparou-se a evolução temporal da ocupação da fila de espera do nó R1, do RTT e da largura de banda utilizada na ligação R1-R2. Estes parâmetros são apresentados nas figuras 6.13, 6.14 e 6.15, respectivamente.

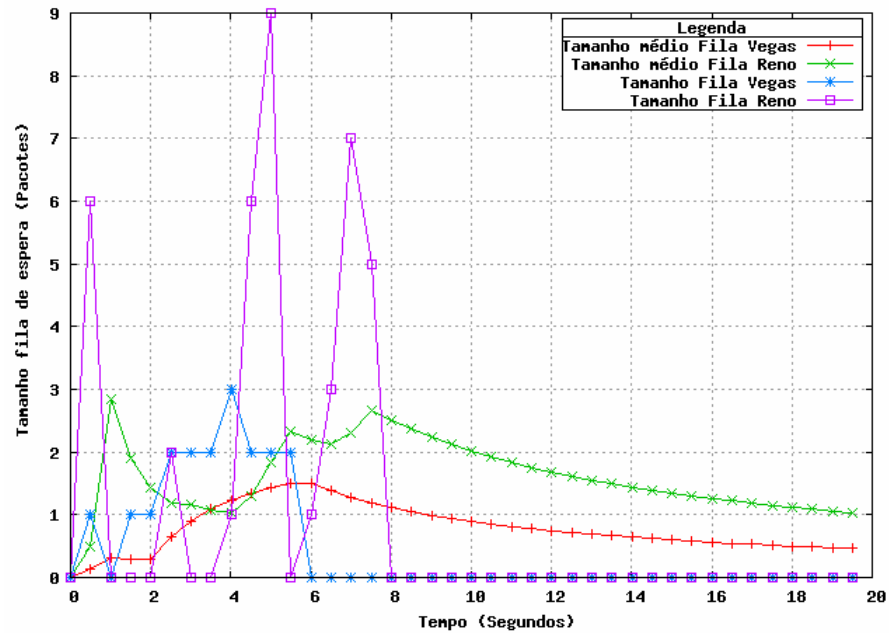


Figura 6.13 – Ocupação da fila de espera, TCP Reno Vs TCP Vegas

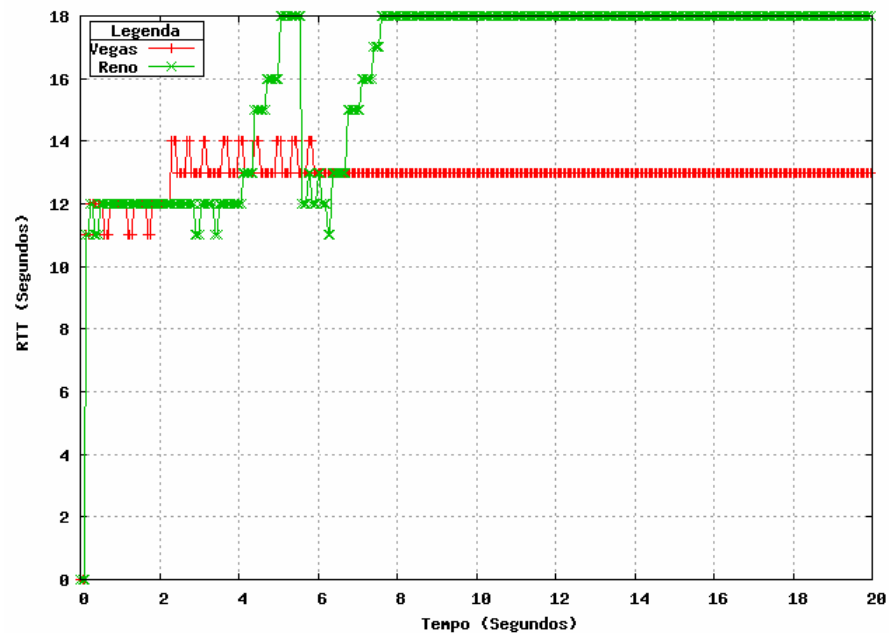


Figura 6.14 – Evolução do valor de RTT, TCP Reno Vs TCP Vegas

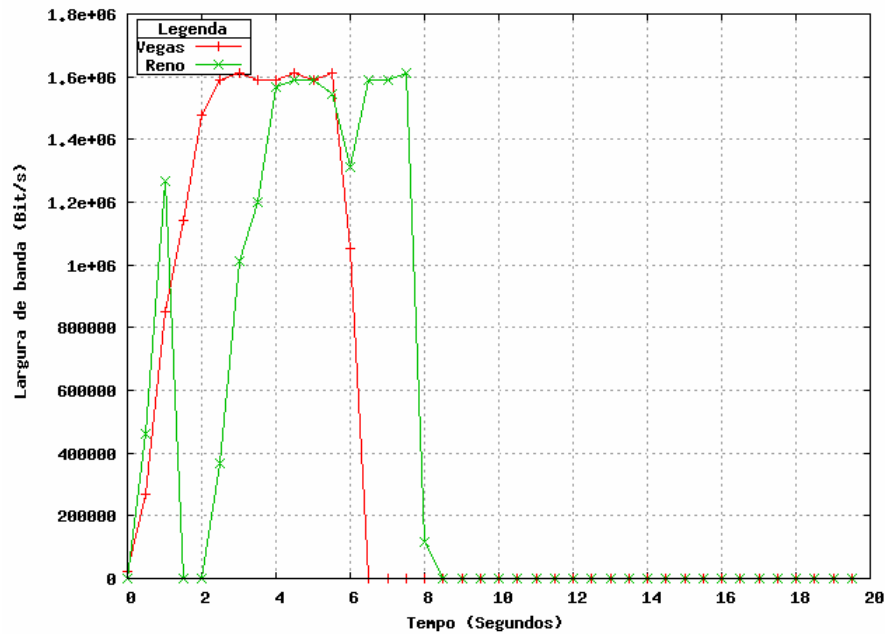


Figura 6.15 – Ocupação da ligação, TCP Reno Vs TCP Vegas

O TCP Vegas tem como objectivo colocar pelo menos α e no máximo β pacotes na fila de espera. Sabendo que $\alpha = 1$ e $\beta = 3$, a linha da ocupação da fila, apresentada na figura 6.13, ilustra essa ocupação. No caso do TCP Reno, a ocupação da fila de espera é sempre utilizada até ao máximo admissível. Quando a fila de espera enche, os pacotes são descartados. A figura 6.13 mostra o efeito do crescimento exponencial da janela de congestionamento do TCP Reno representado pelos picos do tamanho da fila de espera. As constantes actualizações do tamanho da janela provocam oscilações no valor de RTT, como se pode constatar na figura 6.14. No caso do TCP Vegas, os valores de RTT estão bem definidos entre dois valores muito próximos, o que resulta numa menor variação do atraso dos pacotes. Essas oscilações também influenciam o aproveitamento da largura de banda, como se pode constatar na figura 6.15: as quebras na ocupação da largura de banda (no TCP Reno) sucedem após uma quebra na janela de congestionamento. Consequentemente existe um pior aproveitamento da largura de banda comparando com o TCP Vegas.

Influência do atraso de propagação

Esta secção tem por objectivo estudar a influência do atraso de propagação. Estas simulações utilizam o mesmo cenário descrito anteriormente, apenas modificando o valor do atraso de propagação da ligação entre os nós R1 e R2. As tabelas 6.3 e 6.4 apresentam os valores obtidos para cada uma das versões; nestas tabelas são registados os números de pacotes

enviados e retransmitidos, o número de confirmações recebidas (ACKs), a largura de banda utilizada e o instante de saída do último pacote da fila de espera do nó R1.

Atraso da ligação R1 – R2 (ms)	Pacotes Enviados	Pacotes Retransmitidos	ACK Recebidos	LB (Kbits/s)	Fim de dados na fila (s)
2	748	32	728	1287.527	6.50
10	750	34	730	1291.068	6.50
20	753	37	731	1200.560	7.00
30	752	36	733	1203.849	7.00
40	749	33	734	1125.180	7.50
50	742	26	731	1050.583	8.00
60	734	18	726	1043.388	8.00
70	736	20	726	798.883	10.46
80	739	23	727	799.985	10.46
90	733	17	726	883.376	9.46
100	733	17	726	798.883	10.46

Tabela 6.3 – Influência do atraso de propagação no TCP Reno

Atraso da ligação R1 – R2 (ms)	Pacotes Enviados	Pacotes Retransmitidos	ACK Recebidos	LB (Kbits/s)	Fim de dados na fila (s)
2	715	0	715	1454.678	5.50
10	715	0	715	1454.678	5.50
20	715	0	715	1454.678	5.50
30	715	0	715	1454.678	5.50
40	715	0	715	1333.555	6.00
50	715	0	715	1333.555	6.00
60	715	0	715	1231.053	6.50
70	719	4	716	805.545	9.96
80	719	4	716	767.021	10.46
90	719	4	716	700.061	11.46
100	719	4	716	670.782	11.96

Tabela 6.4 – Influência do atraso de propagação no TCP Vegas

Nas tabelas 6.3 e 6.4 verifica-se uma coerência nos resultados obtidos para os 7 primeiros atrasos. No caso do TCP Reno, à medida que o atraso da ligação R1-R2 aumenta, o número de pacotes retransmitidos diminui: por consequência o número de pacotes enviados também diminui. No caso do TCP Vegas, à medida que o atraso da ligação R1-R2 aumenta, não se regista qualquer alteração nos resultados obtidos. Em ambas as tabelas, na passagem dos 60 para 70 ms existe uma diferença de evolução dos resultados. De modo a melhor entender o efeito do atraso de propagação é analisada, nas figuras 6.16 e 6.17, a evolução temporal das respectivas janelas de congestionamento para diversos valores de atrasos de propagação na ligação R1-R2.

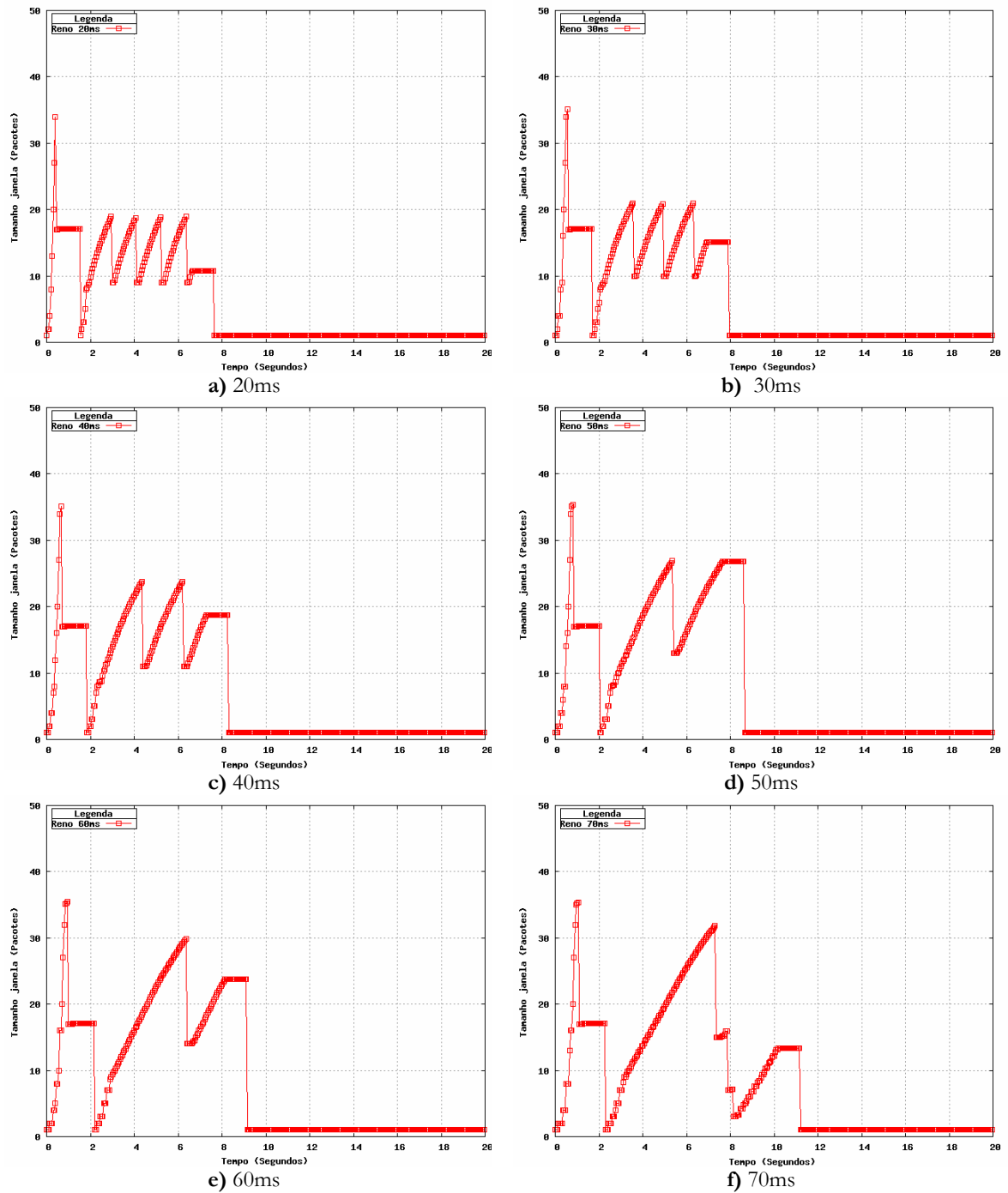


Figura 6.16 – Janela de congestionamento do TCP Reno para diferentes atrasos na ligação R1-R2

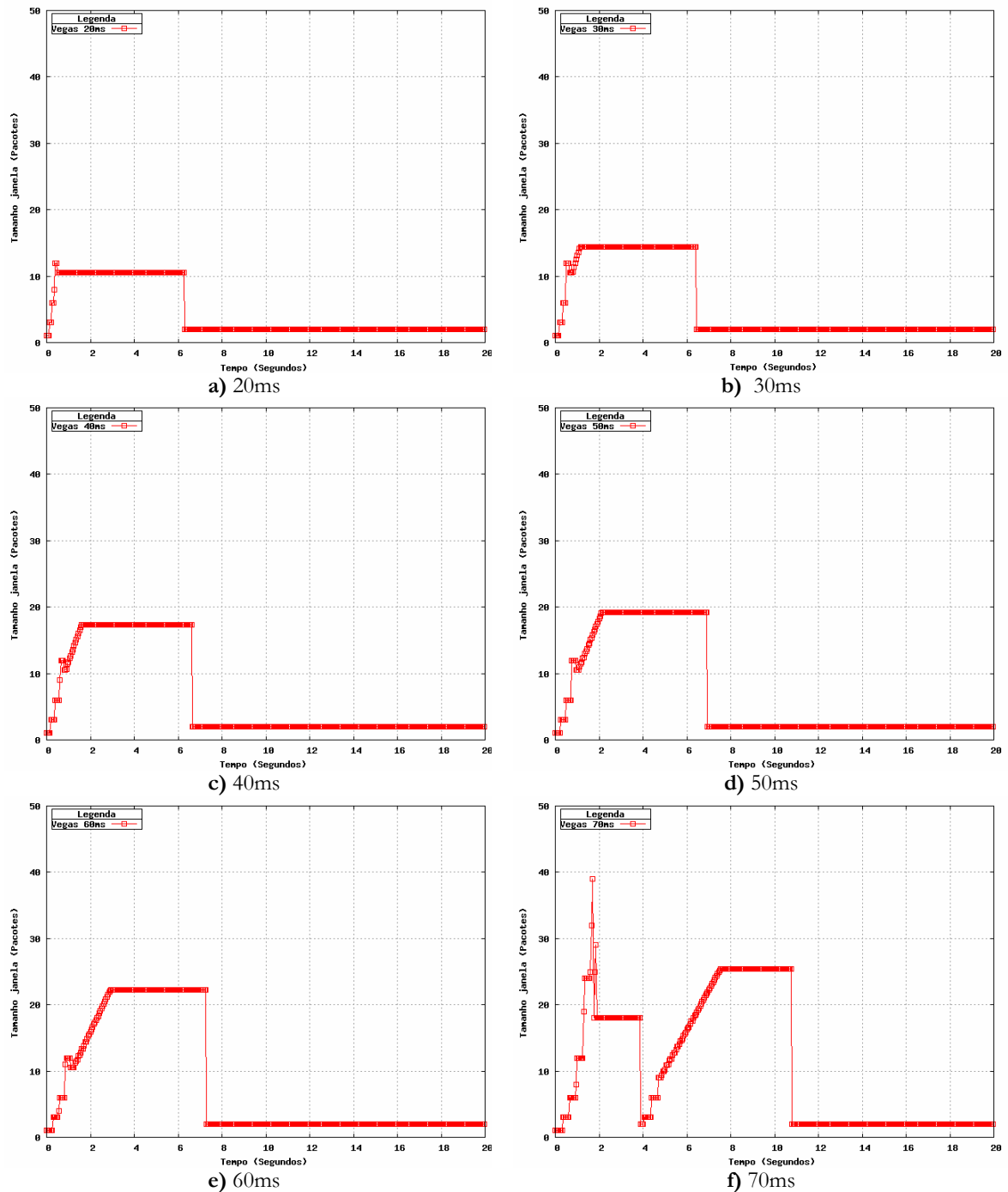


Figura 6.17 – Janela de congestionamento do TCP Vegas para diferentes atrasos na ligação R1-R2

Analisando a evolução temporal das janelas de congestionamento do TCP Reno, depois de ultrapassado o *Slow Start* inicial, verifica-se que à medida que o atraso de propagação aumenta, as fases *Congestion Avoidance* demoram mais tempo e são cada vez em menor número. Relembrando o princípio de funcionamento do mecanismo de congestionamento do TCP Reno, isso deve-se ao facto de o emissor diminuir a sua janela de congestionamento quando

recebe do receptor a informação que um dos seus pacotes foi descartado. O aumento do atraso de propagação faz com que essa mensagem demore mais tempo a chegar.

Como o TCP Vegas estima a largura de banda disponível, através do valor esperado de RTT, ele adapta-se melhor às variações dos atrasos das ligações. Na tabela 6.4 verificava-se uma alteração no comportamento do TCP Vegas na passagem dos 60 para os 70 ms de atraso de propagação, e a partir desse ponto são sempre descartados 4 pacotes. Analisando os gráficos da evolução temporal da janela de congestionamento, alínea f) da figura 6.17, constata-se que esses descartes acontecem na fase *Slow Start* inicial. Na descrição do TCP Vegas foi referido que a fase *Slow Start* inicial tinha sido alterada de modo a evitar perdas de pacotes. No entanto, estes resultados não comprovam essa situação. De modo a entender o que de concreto se está a passar nesta fase, foram elaborados gráficos da evolução da janela de congestionamento e da ocupação da fila de espera para os 4 segundos iniciais e para os seguintes atrasos de propagação: 60 e 70 ms, apresentados nas figuras 6.18 e 6.19.

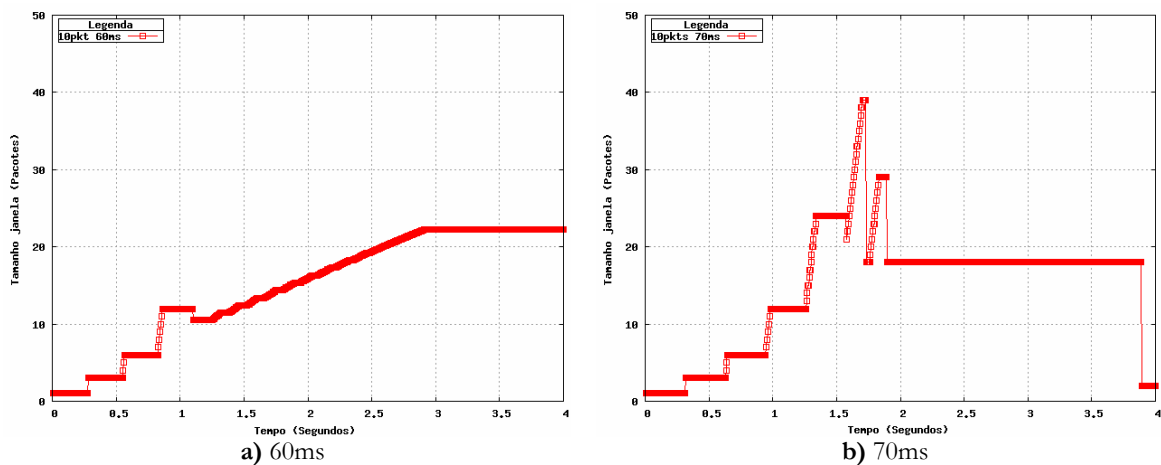


Figura 6.18 – Janela de congestionamento do TCP Vegas para 60 ms e 70 ms

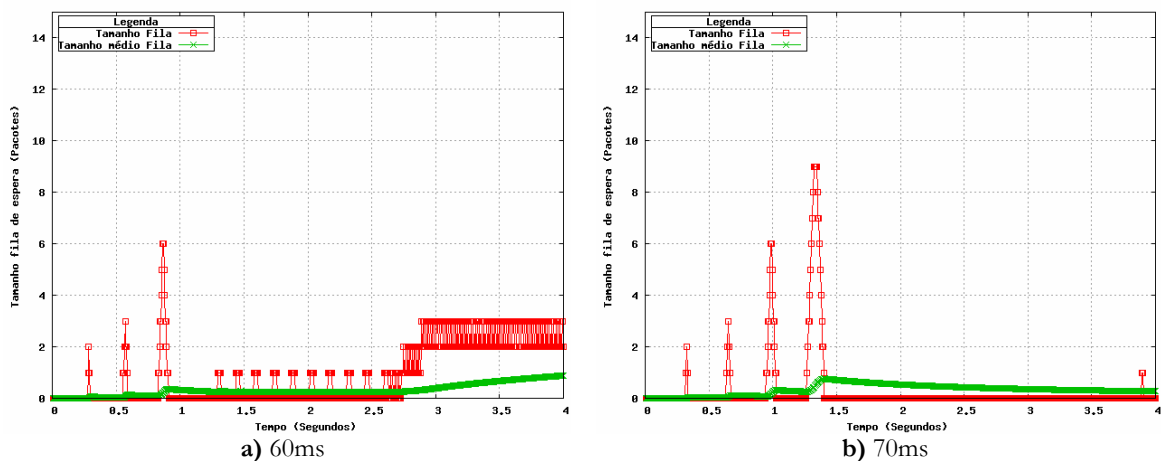


Figura 6.19 – Ocupação da fila de espera do TCP Vegas para 60 ms e 70 ms

Durante a fase *Slow Start* inicial, o TCP Vegas estabiliza o valor da sua janela de congestionamento em RTT alternados, de modo a poder calcular a diferença entre a taxa esperada e a actual. Se esse valor for superior ao valor de γ (1 por defeito), o TCP Vegas comuta para a fase *Congestion Avoidance*. Nas figuras 6.18 a) e b) é bem visível o estabilizar da janela de congestionamento. Analisando a figura 6.18 b), o valor da janela de congestionamento nesses patamares é: 1, 3, 6, 12 e 24 pacotes, respectivamente. Analisando a ocupação da fila de espera, figuras 6.19 a) e b), quando a janela de congestionamento passa do patamar 1 para o 3 são colocados 2 pacotes na fila de espera. Mais uma vez, na passagem do valor 6 para o 12 são colocados 6 pacotes na fila. Na passagem do patamar 12 para o 24 (figura 6.19 b)), a fila de espera tem que ter capacidade para absorver os 12 pacotes, o que não acontece porque o seu limite é de 10 pacotes. Dimensionando a fila de espera com capacidade igual a 13 pacotes eliminam-se as perdas no *Slow Start* inicial como é demonstrado na figura 6.20. Na figura 6.21 são visualizados os 12 pacotes na fila de espera.

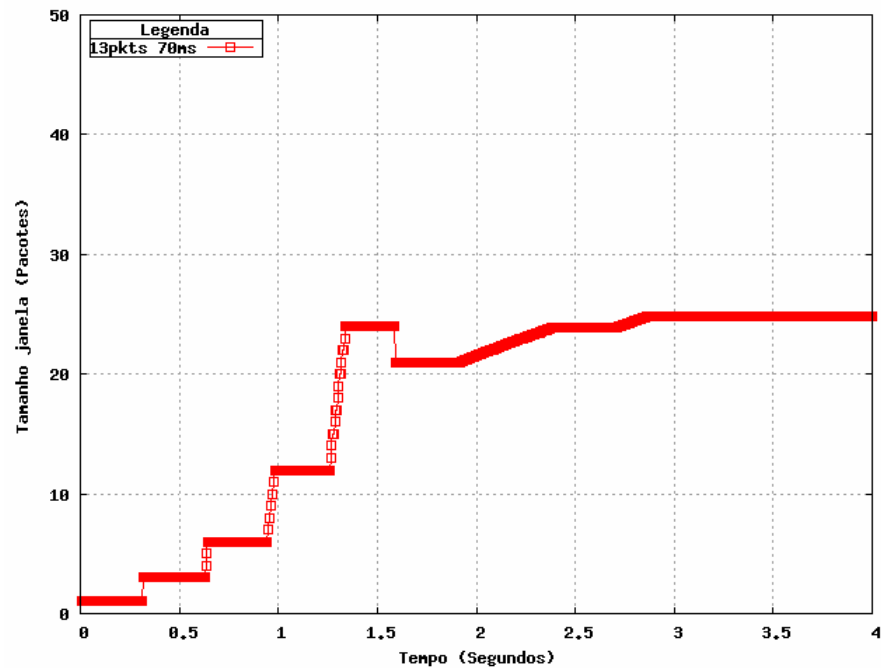


Figura 6.20 – Janela de congestionamento do TCP Vegas para 70 ms com tamanho da fila de 13 pacotes

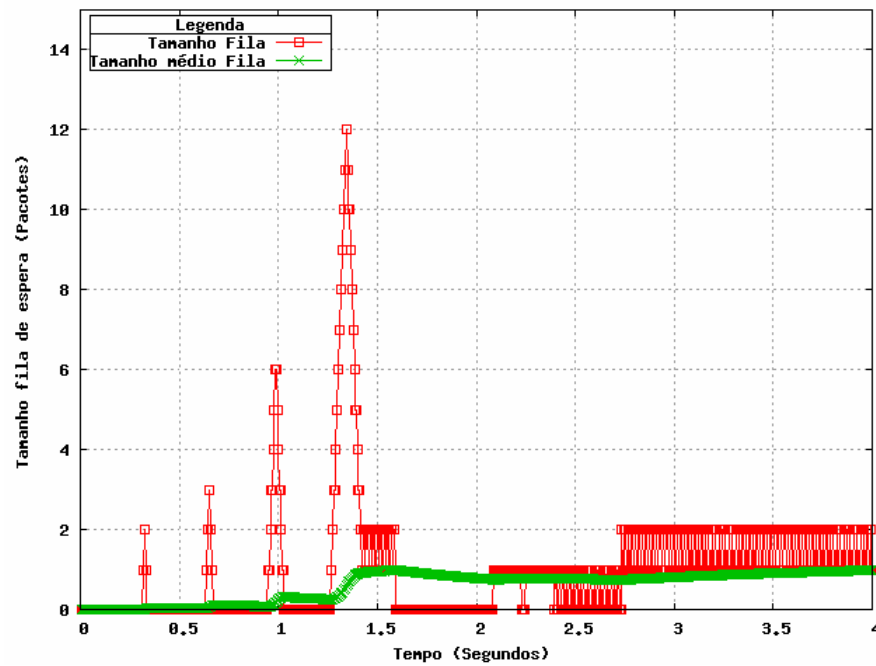


Figura 6.21 – Ocupação da fila do TCP Vegas para 70 ms com tamanho da fila de 13 pacotes

Comparando os gráficos 6.18 a) e b), verifica-se que para o caso dos 60 ms, a fase *Slow Start* é interrompida quando o tamanho da janela de congestionamento é igual a 12; para o caso dos 70 ms, essa mudança só é efectuada quando a janela é igual a 24. A figura 6.22 apresenta a troca de pacotes entre o emissor e o receptor, desde do início da transmissão até a janela de congestionamento do emissor ser igual a 12 pacotes, momento em que para os 60 ms a fase *Slow Start* acaba, o que não acontece para os 70 ms. O objectivo da tabela 6.5 é apresentar os cálculos efectuados durante a troca de pacotes, nos pontos A, B e H, para os atrasos de propagação 60 e 70 ms, de forma a entender o porquê da passagem (ou da não passagem), para a fase *Congestion Avoidance*.

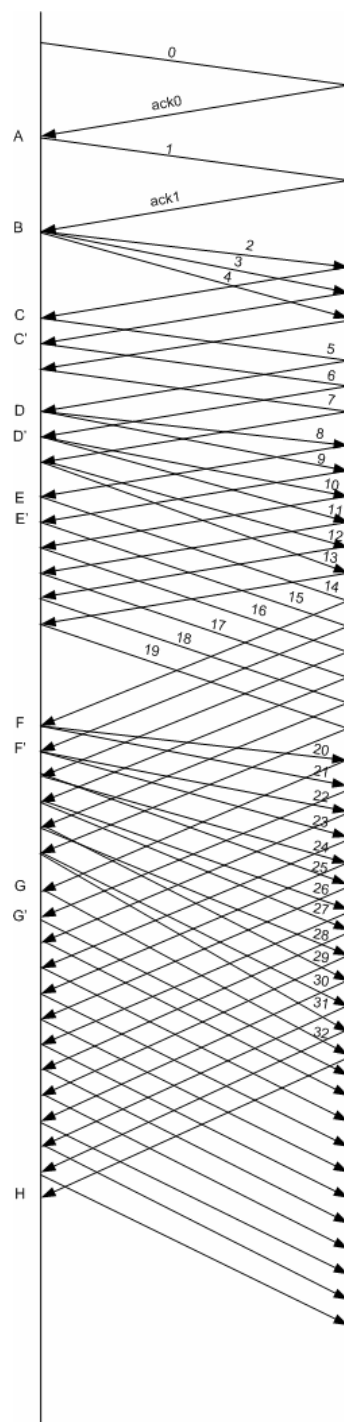


Figura 6.22 – Troca de pacotes entre o emissor e o receptor, até $Cwnd=12$

	60 ms	70 ms
A	<p>Chega ack do 0 $RTT = RTT_0 = 0.137504 = \text{baseRTT}$ $v_sumRTT_ e v_cnt_RTT_ = 0$ $rttLen = 1$ $actual = 7.27 / \text{esperado} = 7.27 / \text{delta} = 0$ Marca próximo = 1</p> <p>Envia 1</p>	<p>$RTT = RTT_0 = 0.157504 = \text{baseRTT}$</p>
B	<p>Chega ack do 1 Actualiza janela = 2 $RTT = RTT_1 = 0.137504$ $v_sumRTT_ e v_cnt_RTT_ = 0$ $rttLen = 1$ $actual = 7.27 / \text{esperado} = 7.27 / \text{delta} = 0$ $\text{delta} < 1$ Marca próximo = 2</p> <p>$Cwnd = Cwnd + 1$ (3) Envia 2, 3 e 4</p>	<p>Chega ack do 1 Actualiza janela = 2 $RTT = RTT_1 = 0.157504$ $v_sumRTT_ e v_cnt_RTT_ = 0$ $rttLen = 1$ $actual = 6.35 / \text{esperado} = 6.35 / \text{delta} = 0$ $\text{delta} < 1$ Marca próximo = 2</p> <p>$Cwnd = Cwnd + 1$ (3) Envia 2, 3 e 4</p>
	<p>○ ○ ○</p>	<p>○ ○ ○</p>
H	<p>Chega ack do 32 $RTT = (RTT_{20} + RTT_{21} + RTT_{22} + RTT_{23} + RTT_{24} + RTT_{25} + RTT_{26} + RTT_{27} + RTT_{28} + RTT_{29} + RTT_{30} + RTT_{31}) / 12 = 0.158504$ $v_sumRTT_ e v_cnt_RTT_ = 0$ $rttLen = 12$ $actual = 75.71 / \text{esperado} = 87.27 / \text{delta} = 2$ $\text{delta} > 1$ FIM SLOW-START</p> <p>Marca próximo = 44</p>	<p>$RTT = 0.178504$</p> <p>$actual = 67.23 / \text{esperado} = 76.19$ $\text{delta} < 1$</p> <p>SÓ NO PRÓXIMO RTT SIM é que comuta de fase</p>

Tabela 6.5 – Cálculos efectuados em alguns pontos da figura 6.31

No ponto A verifica-se que o valor de *BaseRTT* para 60 ms difere de 20 ms em relação ao valor para 70 ms, o que faz sentido já que dos atrasos contabilizados pelo *BaseRTT* (de transmissão, de espera nas filas e de propagação), apenas o de propagação é incrementado de 10 ms em cada percurso (10 ms na ida e 10 ms na volta). O gráfico permite identificar o período de tempo designado por *RTT* sim, onde a cada pacote recebido o emissor envia um pacote; permite também verificar o período de tempo designado por *RTT* não, onde a cada pacote recebido o emissor envia dois pacotes, duplicando o tamanho da janela. Durante todo o processo o emissor calcula o *RTT* de cada pacote recebido. No final do *RTT* sim, o emissor calcula através de uma média ponderada o valor de *RTT* relativo a todos os pacotes enviados durante o *RTT* não, e utiliza esse valor para o cálculo da taxa actual, de modo a decidir se abandona ou não a fase de *Slow Start*. Não são utilizados para esse cálculo os pacotes enviados

durante o RTT sim porque têm o mesmo valor de RTT , que também é igual ao valor de $BaseRTT$. Nesse período, a janela de congestionamento não é incrementada e a rede tem capacidade para transmitir os pacotes sem recorrer a fila de espera. O ponto H da tabela apresenta os cálculos efectuados pelo emissor na decisão de comutar para a fase *Congestion Avoidance* no caso do atraso de propagação igual a 60 ms, e de não comutar no caso do atraso de propagação igual a 70 ms. Para o caso dos 60 ms, o valor de δ é superior a 1, e no caso dos 70 ms o valor de δ continua inferior a 1. No anexo I1 é descrito passo-a-passo a troca de pacotes entre o emissor e o receptor, bem como os cálculos efectuados.

O estudo dos 4 segundos iniciais permite concluir que o valor do atraso dos pacotes é proporcional ao tempo da fase inicial *Slow Start*. Nessa fase e durante o(s) crescimento(s) exponencial(ais) da sua janela de congestionamento em RTT alternados, a fila de espera tem que ter capacidade para armazenar os pacotes gerados. O atraso dos pacotes na rede tem grande impacto no dimensionamento das filas de espera.

Influência de um segundo fluxo de 350KBytes durante a transferência de 1MBytes.

Esta secção tem como objectivo estudar a influência de um segundo fluxo de 350 Kbytes, iniciado 1 segundo depois do primeiro, em que cada um dos fluxos está associado ao protocolo TCP Reno ou Vegas. O cenário inicial é alterado, colocando mais um fluxo (fluxo 2) entre S2 e S4. Alternando o tipo de TCP presente nos agentes de cada fluxo, os resultados de largura de banda utilizados obtidos são apresentados nas tabelas 6.6 a 6.9 e nos gráficos da figura 6.23.

Fluxo	Pacotes Enviados	Pacotes Retransmitidos	ACK Recebidos	LB (Kbits/s)	Fim de dados na fila (s)
1 (TCP Reno)	744	28	731	889.468	9.46
2 (TCP Reno)	252	36	234	315.636	8.50

Tabela 6.6 – TCP Reno (1Mbytes) + TCP Reno (350Kbytes)

Fluxo	Pacotes Enviados	Pacotes Retransmitidos	ACK Recebidos	LB (Kbits/s)	Fim de dados na fila (s)
1 (TCP Reno)	750	34	733	847.108	9.96
2 (TCP Vegas)	215	0	215	601.248	4.00

Tabela 6.7 – TCP Reno (1Mbytes) + TCP Vegas (350Kbytes)

Fluxo	Pacotes Enviados	Pacotes Retransmitidos	ACK Recebidos	LB (Kbits/s)	Fim de dados na fila (s)
1(TCP Vegas)	715	0	715	1000.375	8.00
2 (TCP Reno)	244	28	229	403.825	6.50

Tabela 6.8 – TCP Vegas (1Mbytes) + TCP Reno (350Kbytes)

Fluxo	Pacotes Enviados	Pacotes Retransmitidos	ACK Recebidos	LB (Kbits/s)	Fim de dados na fila (s)
1 (TCP Vegas)	715	0	715	1067.022	7.50
2 (TCP Vegas)	216	1	215	481.119	5.00

Tabela 6.9 – TCP Vegas (1Mbytes) + TCP Vegas (350Kbytes)

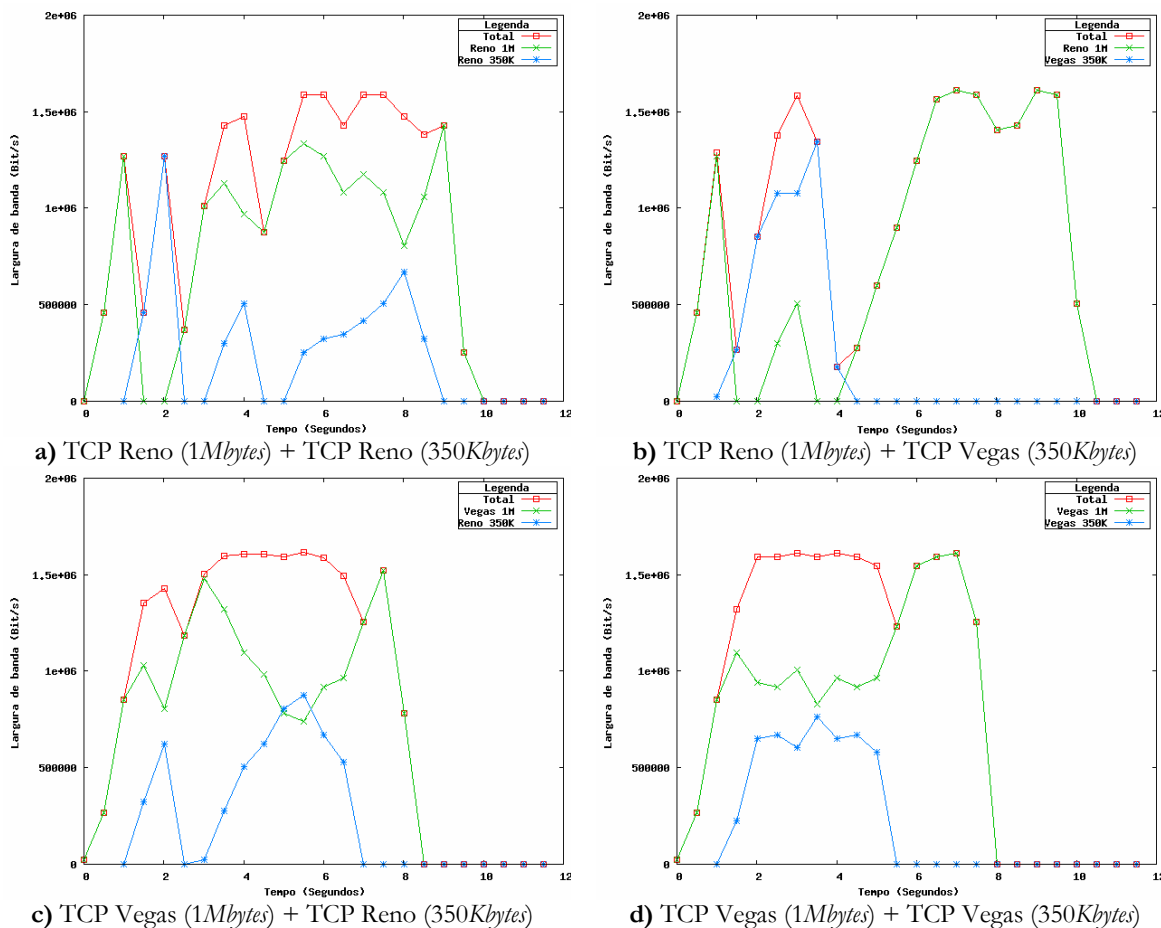


Figura 6.23 – Ocupação da ligação

Analisando os resultados obtidos nas tabelas, verifica-se que o TCP Reno retransmite sempre cerca de 30 pacotes em cada fluxo, enquanto que o TCP Vegas apenas retransmite 1 pacote num dos 4 cenários. Esta diferença do número de pacotes retransmitidos entre cada versão deve-se ao método utilizado para estimar a largura de banda disponível. O TCP Reno utiliza um método mais agressivo e tenta impôr-se na rede. Analisando os gráficos obtidos, conclui-se que o TCP Vegas não se tenta impor, explorando apenas a largura de banda disponibilizada. Nas alíneas b) e c), o TCP Vegas aproveita o *Slow Start* inicial do TCP Reno para utilizar quase toda a largura de banda disponível. Na figura b), quando o TCP Reno começa a estabilizar, o TCP Vegas já praticamente transmitiu toda a sua informação; no caso da figura c) o TCP Reno impõe-se obrigando o TCP Vegas a libertar largura de banda. Na partilha do meio comum o TCP Vegas não influencia o TCP Reno, mas o TCP Reno tenta sempre impor-se,

influenciando o TCP Vegas. Quando ambos os fluxos são do mesmo tipo (figuras a) e d)), no caso do TCP Reno e passado a fase *Slow Start* inicial, os fluxos tendem a partilhar a largura de banda independentemente do instante inicial da transmissão; no caso do TCP Vegas (figura d)), o segundo fluxo estima uma largura de banda disponível inferior ao calculado pelo primeiro fluxo. O cálculo da largura de banda disponível é adaptativo: assim que o segundo fluxo diminui a sua ocupação da largura de banda, o primeiro fluxo (que ainda tem dados para transmitir) aumenta a sua ocupação da largura de banda.

Influência de um segundo fluxo de 1MBytes em simultâneo com a transferência dos 1MBytes iniciais.

Esta secção tem como objectivo estudar a influência de um segundo fluxo que é iniciado simultaneamente com o primeiro. O cenário inicial é alterado, colocando mais um fluxo (fluxo 2) entre S2 e S4. Esse novo fluxo transfere 1Mbytes. Alternando o tipo de TCP presente nos agentes de cada fluxo, os resultados obtidos são apresentados nas tabelas 6.10 a 6.12 e nos gráficos das figuras 6.24 a 6.26.

Fluxo	Pacotes Enviados	Pacotes Retransmitidos	ACK Recebidos	LB (Kbits/s)	Fim de dados na fila (s)
1 (TCP Reno)	749	33	729	647.386	12.96
2 (TCP Reno)	747	31	728	672.450	12.46

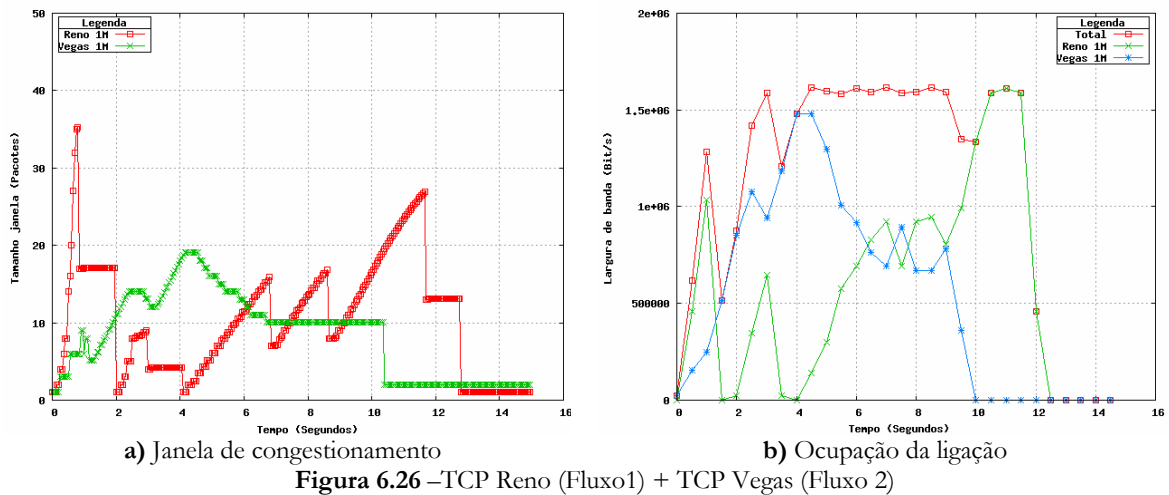
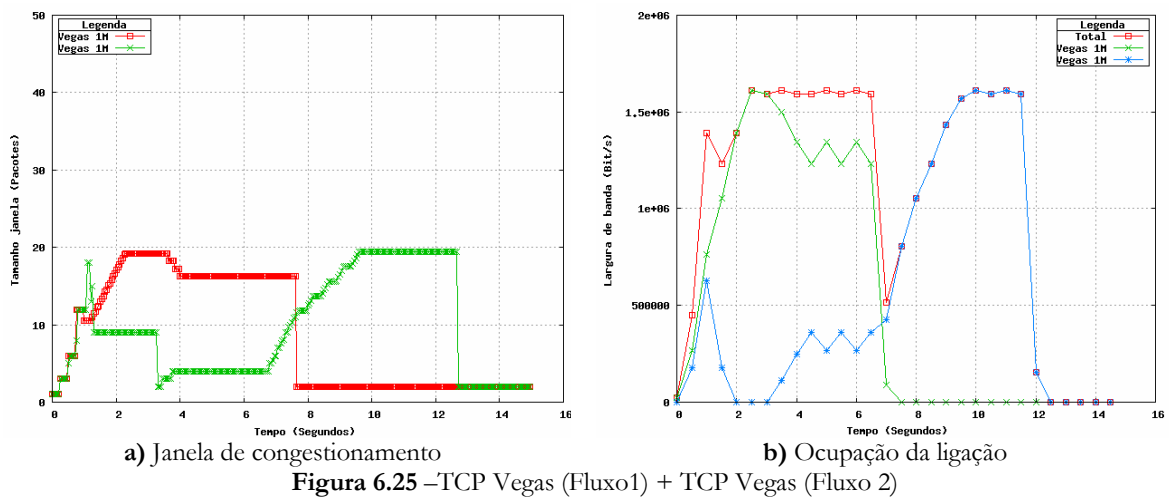
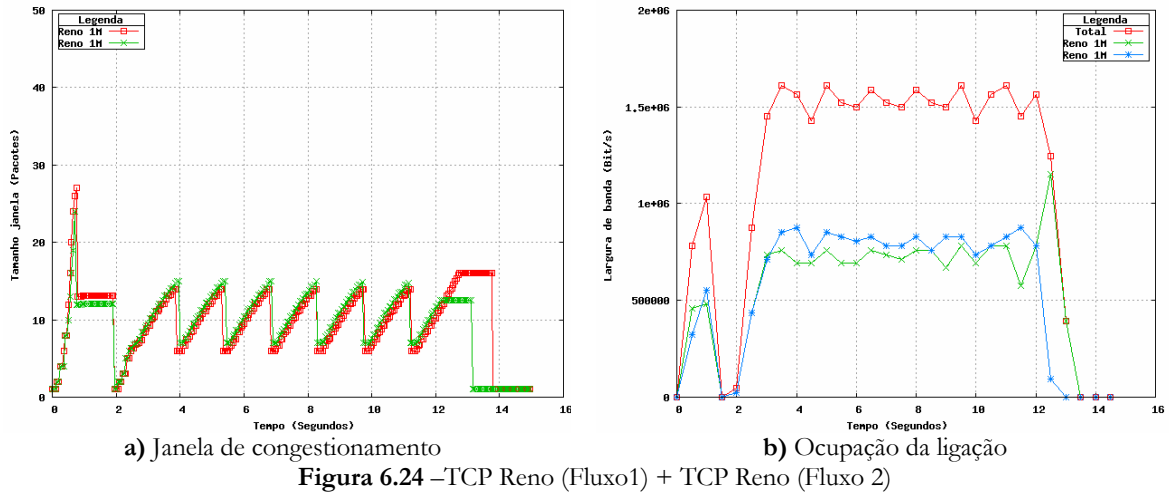
Tabela 6.10 – TCP Reno (1Mbytes) + TCP Reno (1Mbytes)

Fluxo	Pacotes Enviados	Pacotes Retransmitidos	ACK Recebidos	LB (Kbits/s)	Fim de dados na fila (s)
1 (TCP Vegas)	715	0	715	1143.183	7.00
2 (TCP Vegas)	719	4	716	670.782	11.96

Tabela 6.11 – TCP Vegas (1Mbytes) + TCP Vegas (1Mbytes)

Fluxo	Pacotes Enviados	Pacotes Retransmitidos	ACK Recebidos	LB (Kbits/s)	Fim de dados na fila (s)
1 (TCP Reno)	756	40	736	708.282	11.96
2 (TCP Vegas)	717	2	715	846.959	9.46

Tabela 6.12 – TCP Reno (1Mbytes) + TCP Vegas (1Mbytes)



Quando os dois fluxos são do tipo TCP Reno, eles partilham o meio equitativamente. Esta situação é confirmada na tabela 6.10 e na alínea b) da figura 6.24. Quando os dois fluxos são do tipo TCP Vegas, a ligação não é partilhada equitativamente (figura 6.25 b)) e os resultados recolhidos na tabela 6.11 comprovam que o segundo fluxo utiliza cerca de 50 % da largura de

banda utilizada pelo primeiro fluxo. Isso deve-se a um mau dimensionamento da rede: na figura 6.25 a) verifica-se que a capacidade da fila de espera não foi suficiente e o fluxo 2 teve que descartar pacotes, sofrendo de seguida um *timeout*. Esse *timeout* fez com que o valor de *BaseRTT* fosse inicializado. Aos 3 segundos (aproximadamente), quando o fluxo 2 volta a estimar a largura de banda disponível com base nesse valor, o fluxo 2 tem que aguardar que o fluxo 1 acabe de transmitir. Pode-se concluir que os fluxos TCP Vegas iniciados depois (quando já existem fluxos TCP Reno presentes), não irão receber uma porção de tráfego justa, devido ao método conservativo utilizado para estimar a largura de banda.

No caso de um dos fluxos ser TCP Vegas e o outro TCP Reno, constata-se que o TCP Vegas aproveita o *Slow Start* inicial do TCP Reno para ocupar toda a largura de banda disponível (figura 6.26). Assim que o TCP Reno passa a fase *Slow Start* inicial, ele começa a impor-se obrigando o TCP Vegas a ceder largura de banda. O mecanismo do TCP Reno é mais agressivo que o do TCP Vegas na estimação da largura de banda disponível. Essa agressividade faz com que mais pacotes sejam retransmitidos, como se constata nas tabelas 6.10 e 6.12, mas contribui para que o fluxo se imponha face ao TCP Vegas, como se constata no intervalo [4,6] da figura 6.26 b).

Influência do tamanho da fila de espera na partilha TCP Reno e TCP Vegas.

Esta secção tem como objectivo estudar a influência do tamanho da fila de espera quando o TCP Reno e o TCP Vegas partilham a mesma ligação. O cenário inicial é alterado, configurando para cada simulação o tamanho máximo da fila de espera do nó R1. Os resultados obtidos são apresentados na tabela 6.13.

Tamanho da fila do nó R1 (<i>pkts</i>)	Fluxos	Pacotes Enviados	Pacotes Retransmitidos	ACK Recebidos	LB (Kbits/s)	Fim de dados na fila (s)
4	Reno	736	20	721	616.479	13.46
	Vegas	724	9	716	732.013	10.96
7	Reno	742	26	726	670.600	12.46
	Vegas	719	4	716	890.527	9.00
10	Reno	756	40	736	708.283	11.96
	Vegas	717	2	715	846.959	9.46
15	Reno	746	30	732	805.494	10.46
	Vegas	715	0	715	669.845	11.96
25	Reno	718	2	716	1175.892	7.00
	Vegas	715	0	715	699.083	11.46
35	Reno	716	0	716	1266.275	6.50
	Vegas	715	0	715	699.083	11.46
45	Reno	716	0	716	1266.275	6.50
	Vegas	715	0	715	699.083	11.46

Tabela 6.13 – Influência do tamanho da fila de espera, TCP Reno Vs TCP Vegas

Na presença de filas de espera pequenas o TCP Vegas é mais eficaz. À medida que o tamanho da fila de espera aumenta, o TCP Reno consome a maior parte da largura de banda disponível. Para o tamanho da fila de espera de 4 aos 10 pacotes, a largura de banda ocupada é superior para o TCP Vegas; nos outros casos o TCP Reno ocupa mais largura de banda.

O TCP Reno necessita de algum espaço na fila de espera de modo a poder estimar a largura de banda disponível. Quando o tamanho da fila é igual ou superior ao tamanho da janela *Awnd*, o TCP Reno já não retransmite pacotes. O tamanho da janela *Awnd* impõe o número máximo de pacotes que o emissor pode transmitir sem receber as respectivas confirmações; se a fila de espera tem capacidade para armazenar um número igual de pacotes, não haverá necessidade de retransmitir pacotes porque não existirá descarte. No caso do TCP Vegas, o dimensionamento correcto da fila de espera evita a perda de pacotes na fase *Slow Start*; no caso do cenário escolhido o tamanho da fila de espera deverá ser no mínimo de 13 pacotes.

A coexistência de dois fluxos com versões diferentes de TCP (Reno e Vegas) não é impossível, mas é necessário pressupor condições iniciais específicas para que a partilha de ligações entre os dois fluxos seja justa.

6.5 Conclusões

Este capítulo apresentou o estudo de diferentes versões de TCP no sentido de compreender a necessidade dos novos mecanismos que foram desenvolvidos ao longo da evolução do TCP.

Na primeira parte do estudo experimental efectuado foi possível visualizar, através da evolução temporal da janela de congestionamento e da largura de banda utilizada, o efeito dos mecanismos de controlo de congestionamento introduzidos em cada versão do TCP. A necessidade dos mecanismos de controlo de congestionamento é bem visível quando num cenário de uma rede congestionada, a utilização do TCP original (RFC793) não permite a transmissão de 1MBytes de dados nos 20 segundos da simulação. A implementação dos primeiros mecanismos de congestionamento, *Slow Start* e *Congestion Avoidance*, permitiu no mesmo cenário enviar 1 MBytes de dados durante os 20 segundos da simulação. O terceiro mecanismo introduzido no TCP foi o *Fast Retransmit*, permitiu eliminar os tempos mortos provocados pelo método de detectar as perdas de pacote por *timeout*. Com este mecanismo, as perdas são detectadas através de 3 confirmações repetidas, o que permitiu um melhor aproveitamento da largura de banda. O mecanismo *Fast Recovery* surgiu como um

complemento ao mecanismo *Fast Retransmit*: Quando as perdas são detectadas por três confirmações repetidas, isso significa que a rede não está muito congestionada e o valor da janela de congestionamento deve ser reduzido para metade, continuando na fase *Congestion Avoidance*. A última versão de TCP analisada foi o TCP Vegas: esta versão utiliza um método menos agressivo para estimar a largura de banda disponível. O TCP Vegas alterou o mecanismo *Slow Start* das versões anteriores, de modo a eliminar as perdas de pacotes nesta fase. Os mecanismos de controlo de congestionamento utilizados pelo TCP Vegas permitem um melhor aproveitamento da largura de banda disponível.

Na segunda parte deste estudo experimental comparou-se o TCP Reno com o TCP Vegas, alterando as condições iniciais do cenário. Através da análise da ocupação da fila de espera foi visualizado o princípio de funcionamento do TCP Vegas, que tenta colocar um valor controlado de pacotes na fila de espera. No caso do TCP Reno, a ocupação da fila de espera é sempre utilizada até ao máximo admissível, o que ilustra o comportamento agressivo do TCP Reno na fase de *Congestion Avoidance*.

A influência do atraso de propagação foi analisada para ambas as versões. Esse estudo desencadeou o estudo da fase *Slow Start* inicial na versão TCP Vegas que apresentou resultados diferentes dos esperados. O conceito teórico da versão TCP Vegas, que anuncia o não descarte de pacotes na fase *Slow Start* inicial, só é válido se as filas de espera presentes forem correctamente dimensionadas, com capacidade para armazenar os pacotes gerados na fase do crescimento exponencial em RTT alternados.

A concorrência entre fluxos com diferentes versões de TCP, Reno e Vegas, foi também analisada. Nestes cenários verificou-se que na partilha do meio comum, o TCP Vegas não influencia o TCP Reno, mas por seu lado o TCP Reno tenta sempre impor-se.

Por último foi analisada a influência do tamanho da fila de espera na partilha do meio entre o TCP Vegas e o TCP Reno. Na presença de filas pequenas de espera, o TCP Vegas é mais eficiente, devido ao seu método mais eficaz de estimar a largura de banda disponível. Quando o tamanho da fila de espera é igual ou superior ao tamanho da janela *Window*, já não existe descarte de pacotes no TCP Reno. A coexistência destas duas versões de TCP deve pressupor um bom dimensionamento da rede de modo a que a partilha do meio seja justa.

Em cenários bem dimensionados, a escolha sobre a versão de TCP a utilizar deverá recair sobre o TCP Vegas, que permite um melhor aproveitamento da largura de banda e uma não influência para os restantes fluxos. Em cenários em que a largura de banda tem que ser conquistada, o TCP Reno será a melhor opção. O seu método mais agressivo para estimar a largura de banda disponível, ajudará a garantir uma fracção da largura de banda disponível.

7. Mecanismos de policiamento

O controlo de fluxos fornece mecanismos de controlo da taxa de transmissão do tráfego que entra na rede. Estes mecanismos podem ser de policiamento ou de formatação de tráfego dependendo das necessidades de configuração da rede. Ambos utilizam métodos semelhantes na classificação e medição do tráfego que viola os parâmetros pré-definidos, classificando os pacotes respectivos como fora do perfil acordado com a rede (*out-of-profile*). Os mecanismos diferem no modo como reagem a essas violações: o policiamento tipicamente descarta os pacotes classificados como *out-of-profile* ou marca-os como *out-of-profile* para que nos nós seguintes esses pacotes sofram um tratamento distinto (por exemplo atribuindo uma maior probabilidade de descarte), enquanto que o formatador de tráfego atrasa os pacotes em excesso utilizando filas de espera e um servidor com uma determinada taxa de transmissão. Neste capítulo são analisados os mecanismos de policiamento.

O mecanismo de policiamento mais conhecido em redes de pacotes é o *Token bucket*. O IETF definiu alguns algoritmos para policiamento, como por exemplo o srTCM [29] (*Single Rate Three Color Marker*), trTCM [30] (*Two Rate Three Color Marker*) e tswTCM [31] (*Time Sliding Window Three Colour Marker*). Os dois primeiros são baseados no *Token bucket* e o último baseado na estimação da taxa média. O mecanismo de policiamento utiliza cores para marcar os pacotes, usualmente o verde, o amarelo e o vermelho. Essas cores correspondem, no modelo PHB AF da arquitectura DiffServ, descrito no capítulo 8, a um determinado DSCP (*DiffServ Code Point*) associado a um nível de descarte.

Este capítulo está organizado da seguinte forma. A secção 7.1 descreve o *Token bucket*. A secção 7.2 descreve o srTCM. O trTCM é descrito na secção 7.3. O tswTCM é descrito na secção 7.4. As simulações efectuadas neste capítulo são apresentadas na secção 7.5. As conclusões deste capítulo são descritas na secção 7.6.

7.1 *Token bucket*

A ideia principal deste algoritmo tem como base um balde (*bucket*) para onde são depositados talões (*tokens*), gerados em cada ΔT segundos. Este algoritmo possui dois parâmetros configuráveis, o CBS (*Committed Burst Size*) que representa o tamanho do balde em *bytes* e a taxa de preenchimento do balde em *bits/s* designada CIR (*Committed Information Rate*). Este

algoritmo utiliza apenas duas cores, marcando os pacotes com a cor verde (*in-profile*) ou com a cor vermelha (*out-profile*).

O policiamento é feito do seguinte modo (figura 7.1). Se existirem talões no balde os pacotes são marcados com a cor verde, caso contrário são marcados com a cor vermelha. Para que um pacote seja marcado com a cor verde, deve existir no balde pelo menos um número de *bytes* em talões igual ao tamanho em *bytes* do pacote. Neste caso, o pacote captura os talões (eliminando-os do balde) e o pacote é transmitido e marcado com a cor verde. Quando não existem talões no balde, os pacotes são transmitidos e marcados com a cor vermelha. Quando a taxa de chegada de pacotes ao *Token bucket* é inferior à taxa de geração de talões (CIR), os talões são acumulados no balde. Se o balde encher os talões são descartados.

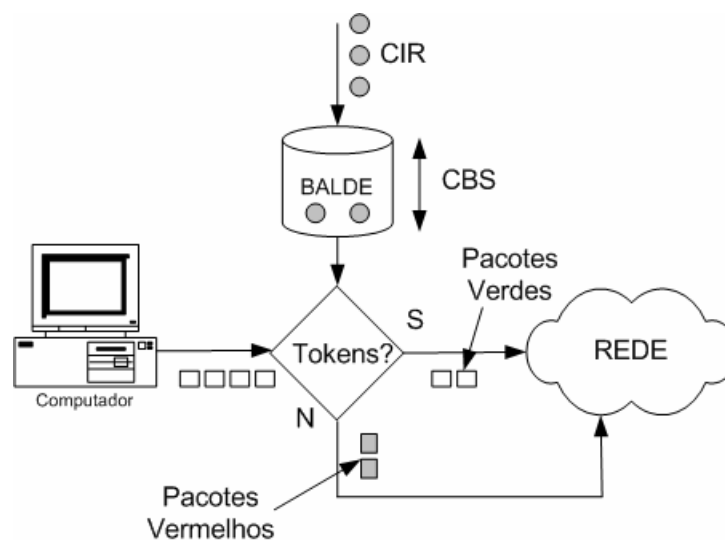


Figura 7.1 – *Token bucket*

O *Token bucket* é um algoritmo que possibilita a transmissão de rajadas de pacotes, onde o tamanho máximo da rajada de pacotes classificados *in-profile* é determinado pelo tamanho do balde (CBS) e a taxa de saída média dos pacotes marcados a verde é igual à taxa de enchimento do balde (CIR).

7.2 Single Rate Three Color Marker

O srTCM é um mecanismo de policiamento baseado em 2 baldes, que recebem talões à mesma taxa CIR. O tamanho (em *bytes*) do primeiro balde é igual a CBS e do segundo balde é igual a EBS (*Excess Burst Size*). Este mecanismo marca os pacotes com 3 cores diferentes (verde, amarelo e vermelho). A figura 7.2 apresenta o princípio de funcionamento do srTCM.

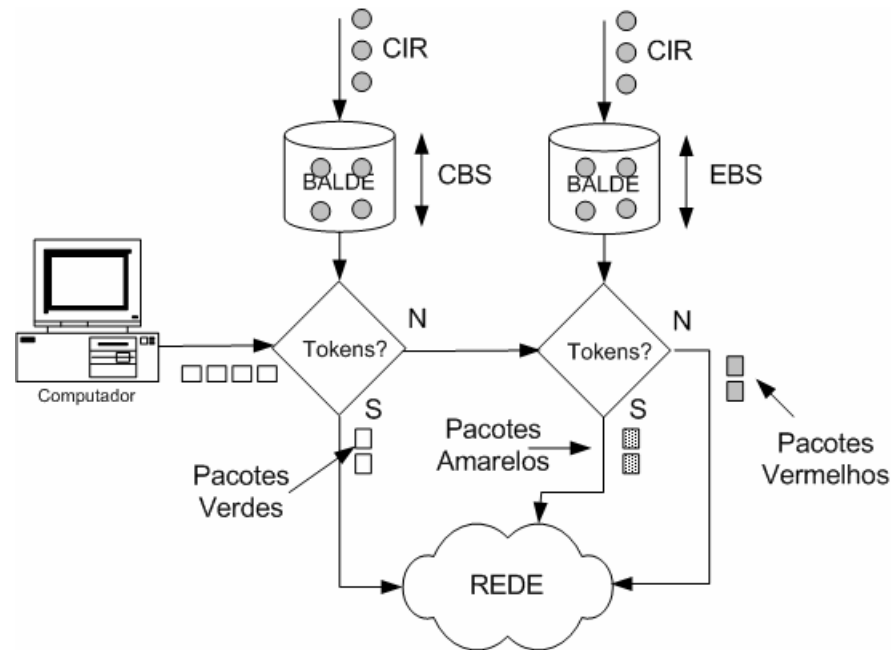


Figura 7.2 – srTCM

Considerando que os pacotes são de tamanho fixo e igual ao tamanho dos talões, quando um pacote chega e existe 1 talão no primeiro balde, esse pacote é marcado com a cor verde e o número de talões no primeiro balde é decrementado de 1 unidade. Se não existirem pacotes no primeiro balde o pacote é encaminhado para o segundo. O pacote é marcado a amarelo se existir 1 talão no segundo balde e o número de talões no segundo balde é decrementado de 1 unidade. Se não existirem talões nos dois baldes, o pacote é marcado com a cor vermelha.

A taxa de geração de pacotes (CIR) representa a taxa média negociada para a saída de pacotes marcados com a cor verde e o tamanho do primeiro balde (CBS) representa o tamanho máximo acordado para uma rajada de pacotes também marcada com a cor verde. O tamanho do segundo balde (EBS) representa o tamanho extra da rajada tolerável para pacotes marcados com a cor amarela. Os pacotes que não respeitam a CIR, o CBS e o EBS são marcados de vermelho.

Na prática, os pacotes podem ter tamanho variável, quando um pacote de tamanho B bytes chega, é comparado o tamanho em bytes do pacote com o número de bytes que está no balde se existirem pelo menos B bytes de talões a comparação é verdadeira.

7.3 Two Rate Three Color Marker

O trTCM é semelhante ao mecanismo anterior, com a diferença de que o primeiro balde recebe talões a uma taxa PIR (*Peak Information Rate*) e o segundo balde recebe talões a uma taxa CIR. O número máximo de talões do primeiro balde é PBS (*Peak Burst Size*) e do segundo balde é CBS. Este mecanismo marca os pacotes com 3 cores diferentes (verde, amarelo e vermelho). A figura 7.3 ilustra o princípio de funcionamento do trTCM.

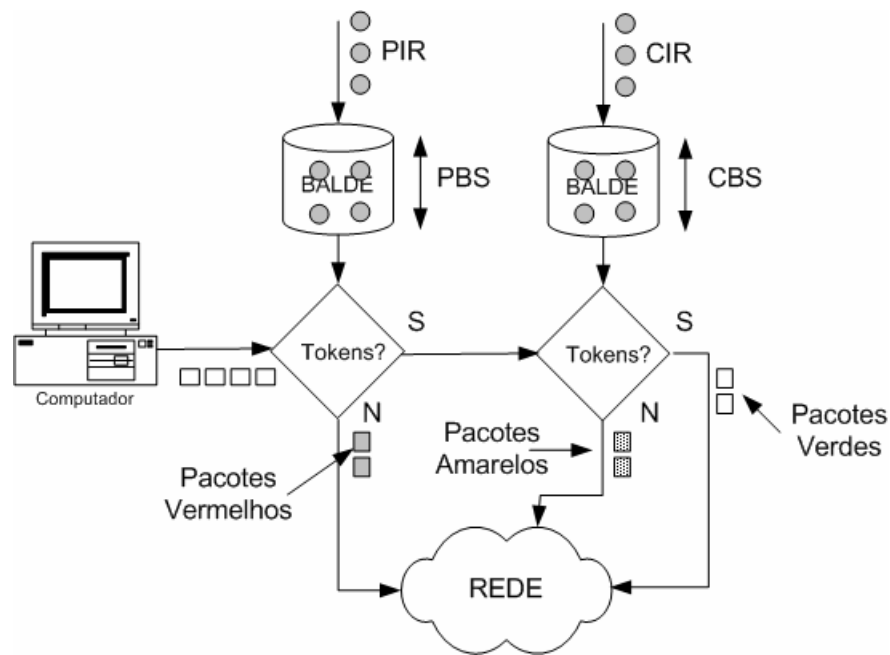


Figura 7.3 – trTCM

Considerando que os pacotes são de tamanho fixo e igual ao tamanho dos talões, quando um pacote chega e não existe 1 talão no primeiro balde, esse pacote é marcado com a cor vermelha. Se existirem talões no primeiro balde o pacote é encaminhado para o segundo. O pacote é marcado a amarelo se não existir 1 talão no segundo balde e o número de talões no primeiro balde é decrementado de 1 unidade. Se existirem talões nos dois baldes, o pacote é marcado com a cor verde e o número de talões de cada balde é decrementado de 1 unidade.

A taxa de geração de pacotes (CIR) representa a taxa média negociada para a saída de pacotes marcados com a cor verde e o tamanho do segundo balde (CBS) representa o tamanho máximo acordado para uma rajada de pacotes marcada com a cor verde. PBS é a rajada máxima aceite pela rede para ser transmitida, marcando os pacotes com a cor amarela, num determinado período a uma taxa PIR (taxa máxima de pico). Os pacotes que não violam o

PIR, PBS, CIR e CBS são marcados de verde. Os pacotes que não violam o PIR e PBS, mas violam CIR e/ou CBS são marcados a amarelo. Os pacotes que violam o PIR e PBS são marcados de vermelho.

Na prática, os pacotes podem ter tamanho variável, quando um pacote de tamanho B bytes chega, é comparado o tamanho em bytes do pacote com o número de bytes que está no balde se existirem pelo menos B bytes de talões a comparação é verdadeira.

7.4 Time Sliding Window Three Colour Marker

O tswTCM é baseado na estimação da taxa média da informação enviada pelos fluxos ou agregados. Este mecanismo pode ser configurado de modo a permite abranger as rajadas de tráfego. A marcação dos pacotes é realizada comparando a taxa de um tráfego de entrada com as taxas CTR (*Committed Target Rate*) e PTR (*Peak Target Rate*) predefinidas, sendo CTR a taxa média contratada e PTR a máxima taxa de informação contratada aceite pela rede por um determinado período. A figura 7.4 apresenta o diagrama de blocos do tswTCM. O tswTCM consiste em 2 componentes, um estimador da taxa e um marcador.

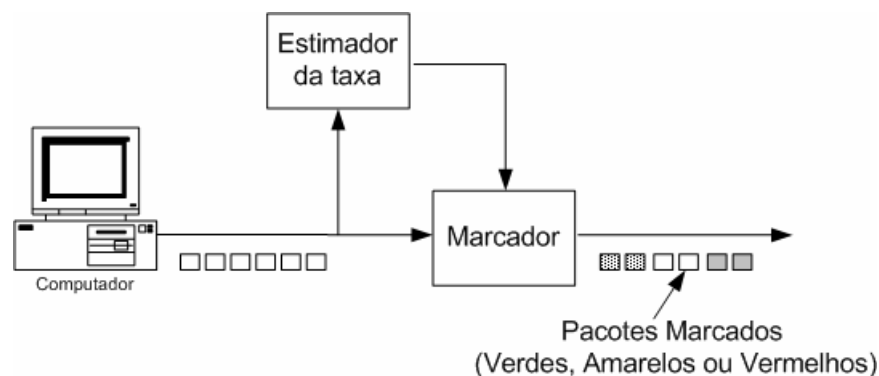


Figura 7.4 – tswTCM

O primeiro bloco mede a taxa média estimada (TME) à chegada de cada novo pacote. Essa taxa é calculada numa janela de tempo finita (AVG_win). O algoritmo utilizado pode ser por exemplo o *time-based rate estimation* descrito em [34]. Inicialmente a TME é igual a CTR, AVG_win é uma constante predefinida e a variável *clock* é igual a 0. As expressões 7.1 a 7.4 são as operações efectuadas para o cálculo da TME no *time-based rate estimation* à chegada de um novo pacote.

$$Bytes_in_win = TME \times AVG_win \quad (7.1)$$

$$New_bytes = Bytes_in_win + size_pkt \quad (7.2)$$

$$TME = New_bytes / (clock_pkt - clock + AVG_win) \quad (7.3)$$

$$clock = clock_pkt \quad (7.4)$$

Nestas equações a variável *clock_pkt* representa o instante de chegada do pacote e a variável *size_pkt* representa o tamanho do pacote em *bytes*. Na expressão 7.1 é calculado o número de *bytes* durante o intervalo de tempo *AVG_win* à taxa TME calculada anteriormente. Ao número de *bytes* calculado é incrementado na expressão 7.2 o tamanho do pacote em *bytes*. A expressão 7.3 recalcula o valor de TME utilizando o novo valor de *bytes* calculado, o tamanho da janela (*AVG_win*), o instante de chegada do pacote (*clock_pkt*) e o instante de chegada do pacote anterior (*clock*). O instante de chegada do anterior pacote é actualizado na expressão 7.4, com o valor do instante de chegada do novo pacote.

O marcador marca os pacotes de verde, amarelo e vermelho. A TME é utilizada pelo marcador para determinar com que cor um determinado pacote deve ser marcado. Se TME é menor ou igual a CTR, o pacote é marcado com a cor verde. Se TME é maior do que CTR e menor ou igual a PTR, então os pacotes são marcados a amarelo com um probabilidade P0 (expressão 7.5) e marcados de verde com uma probabilidade (1-P0). Se TME é maior do que PTR, os pacotes são marcados com a cor vermelha com uma probabilidade P1 (expressão 7.6), a amarelo com probabilidade P2 (expressão 7.7), e de verde com probabilidade (1-(P1+P2)).

$$P0 = \frac{(TME - CTR)}{TME} \quad (7.5)$$

$$P1 = \frac{(TME - PTR)}{TME} \quad (7.6)$$

$$P2 = \frac{(PTR - CTR)}{TME} \quad (7.7)$$

O tswTCM pode ser configurado para trabalhar apenas com uma taxa. Se a taxa CTR for igual à taxa PTR, todos os pacotes serão marcados com as cores verde ou vermelho, não existindo pacotes amarelos. Se PTR for configurado à taxa da ligação e CTR for inferior a PTR, todos os pacotes serão marcados de verde ou amarelo, não existindo pacotes vermelhos.

7.5 Simulações

O objectivo desta secção é estudar por simulação os mecanismos de policiamento apresentados nas secções anteriores. O cenário escolhido é apresentado na figura 7.5.

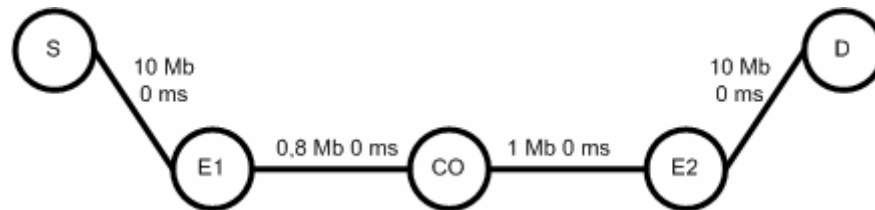


Figura 7.5 – Cenário para simular mecanismos de policiamento

Cada simulação tem a duração de 80 segundos (T_s). Todas as filas de espera são configuradas com capacidade de 50 pacotes. Entre o nó origem S e o nó destino D é estabelecido um fluxo de tráfego, com pacotes de tamanho igual a 100 *bytes* (L_{byte}), intervalo entre chegadas exponencialmente distribuído, taxa de 1000 *Kbps* (T_x) durante 80 segundos e um tempo de *burst* de 500 ms (rajada de 625 pacotes). No nó E1 é colocado em cada simulação um dos mecanismos de posicionamento descritos. Na tabela 7.1 são apresentados os valores configurados para cada mecanismo.

	CIR	CBS	EBS	PIR	PBS	CTR	PTR
Token bucket	100000	2000	X	X	X	X	X
srTCM	100000	2000	3000	X	X	X	X
trTCM	100000	2000	X	200000	3000	X	X
tswTCM	X	X	X	X	X	100000	200000

Tabela 7.1 – Valores configurados em cada mecanismos de policiamento

Os resultados teóricos para cada um dos mecanismos de policiamento são apresentados na tabela 7.2. A primeira coluna representa o número de pacotes que chegaram ao nó E1. As restantes colunas apresentam o número de pacotes marcados (verde, amarelo ou vermelho), que foram enviados ou descartados.

	Recebidos $P_{recebidos}$	VERDE		AMARELO		VERMELHO	
		Enviados P_{verde_e}	Descartados P_{verde_d}	Enviados $P_{amarelo_e}$	Descartados $P_{amarelo_d}$	Enviados $P_{vermelho_e}$	Descartados $P_{vermelho_d}$
Token bucket	100000	10020	0	X	X	69980	20000
srTCM	100000	10020	0	30	0	69950	20000
trTCM	100000	10020	0	10010	0	59970	20000
tswTCM	100000	>10000	0	>10000	0	<60000	20000

Tabela 7.2 – Número teórico de pacotes recebidos e marcados no nó 1

O número de pacotes recebidos, para qualquer dos mecanismos é calculado utilizando a expressão

$$P_{recebidos} = \frac{T_X \times T_S}{8 \times L_{byte}} \quad (7.8)$$

O número de pacotes marcados a verde, para os três primeiros mecanismos é igual

$$P_{verde} = \frac{CIR \times T_S}{8 \times L_{byte}} + \frac{CBS}{L_{byte}} \quad (7.9)$$

Esta expressão representa a soma do número de pacotes que respeitam a taxa CIR com o tamanho máximo da rajada permitida (como existe uma rajada de pacotes durante 500 ms). O tswTCM marca de verde os pacotes que respeitam a taxa CTR (~10000 pacotes), marca mais alguns pacotes com uma probabilidade (1-P0) quando TME é superior a CTR e menor ou igual a PTR, e marca mais alguns pacotes com uma probabilidade (1-(P1+P2)) quando TME é superior a PTR.

O número de pacotes que o srTCM marca a amarelo é igual

$$P_{amarelo} = \frac{EBS}{L_{byte}} \quad (7.10)$$

O número de pacotes que o trTCM marca a amarelo é igual

$$P_{amarelo} = \frac{(PIR - CIR) \times T_S}{8 \times L_{byte}} + \frac{(PBS - EBS)}{L_{byte}} \quad (7.11)$$

Esta expressão representa a soma do número de pacotes entre o valor de PIR e de CIR com o tamanho máximo da rajada permitida. O tswTCM marca a amarelo com uma probabilidade (P0) os pacotes gerados quando TME é superior a CTR e menor ou igual a PTR (~10000 pacotes), e marca mais alguns pacotes com uma probabilidade (P2) quando TME é superior a PTR.

Para os 4 mecanismos, o número de pacotes marcados de vermelho e descartados é

$$P_{vermelho_d} = \frac{(T_X - LB) \times T_S}{8 \times L_{byte}} \quad (7.12)$$

onde LB representa a largura de banda da ligação. No caso dos 3 primeiros mecanismos de policiamento, o número de pacotes marcados a vermelho (e enviados) é igual

$$P_{\text{vermelho}_e} = P_{\text{recebidos}} - P_{\text{verde}} - P_{\text{amarelo}} - P_{\text{vermelho}_d} \quad (7.13)$$

O mecanismo tswTCM marca a vermelho com uma probabilidade (P_1) os pacotes gerados quando TME é superior a PTR (~60000 pacotes). Os valores teóricos para o caso do mecanismo tswTCM são aproximações, porque os valores exactos são de difícil cálculo devido ao modo interativo utilizado pelo mecanismo para obter os valores das probabilidades P_0 , P_1 e P_2 .

A tabela 7.3 apresenta os resultados obtidos por simulação para cada um dos mecanismos de policiamento. Os pacotes descartados são apresentados sob a forma de uma soma, em que a primeira parcela representa os pacotes descartados devido à técnica de descarte DropTail e a segunda parcela devido à técnica de descarte RED (ver capítulo 5).

	Recebidos	VERDE		AMARELO		VERMELHO	
		Enviados	Descartados	Enviados	Descartados	Enviados	Descartados
Token bucket	99999	10018	1+0	X	X	69995	4063+15922
srTCM	99999	10019	0+0	30	0+0	69884	141+19925
trTCM	99999	10019	0+0	10010	0+0	59941	119+19910
tswTCM	99999	10219	0+0	10165	0+6	59541	199+19869

Tabela 7.3 – Número de pacotes recebidos e marcados no nó 1

Os valores obtidos por simulação são aproximadamente iguais aos valores calculados teoricamente. A capacidade da ligação é a mesma para os 4 mecanismos e o número de pacotes descartados deve-se à limitação física dessa ligação, portanto não depende do mecanismo escolhido. Os 3 primeiros mecanismos utilizam o mesmo princípio para determinar o número de pacotes marcados a verde, obtendo assim o mesmo resultado nas simulações. O mecanismo tswTCM utiliza outro método, que lhe permite marcar mais pacotes a verde que os restantes mecanismos.

No cenário de simulação, os parâmetros CTR e PTR foram alterados de modo a verificar os 2 casos particulares de configuração do mecanismo de policiamento tswTCM enunciados na secção 7.4. Os resultados obtidos são apresentados na tabela 7.4.

	Recebidos	VERDE		AMARELO		VERMELHO	
		Enviados	Descartados	Enviados	Descartados	Enviados	Descartados
CTR=100000 PTR=200000	99999	10219	0+0	10165	0+6	59541	199+19869
CTR=100000 PTR=100000	99999	10349	0+0	X	X	69609	75+19966
CTR=100000 PTR=1000000	99999	10346	8+0	69664	3977+16004	X	X

Tabela 7.4 – Número de pacotes recebidos e marcados com tswTCM

Os resultados obtidos por simulação confirmam que se a taxa CTR for igual à taxa PTR, todos os pacotes serão marcados com as cores verde ou vermelho, não existindo pacotes amarelos; se PTR for configurado à taxa da ligação e CTR for inferior a PTR, todos os pacotes serão marcados a verde ou amarelo, não existindo pacotes vermelhos. Dependendo da configuração escolhida é possível utilizar este mecanismo para marcar pacotes com 2 ou 3 cores.

7.6 Conclusões

Neste capítulo foram estudados e analisados 4 mecanismos de policiamento, iniciando o estudo pelo mais conhecido, o *Token bucket*, passando pelo srTCM e trTCM, e finalmente o tswTCM. A principal função do mecanismo de policiamento é marcar os pacotes consoante os parâmetros configurados, tais como a taxa média contratada ou a rajada máxima de pacotes aceite. A escolha do mecanismo para controlar o fluxo permite marcar os pacotes com 2 ou 3 cores, dependendo do número de parâmetros analisados pelos mecanismos.

Os resultados obtidos por simulação permitem validar, comparando com os resultados obtidos teoricamente, as expressões teóricas utilizadas para o cálculo do número de pacotes marcados nos 3 primeiros métodos. Essas expressões exemplificam o princípio de funcionamento de cada mecanismo. Nos dois mecanismos que analisam duas taxas, o trTCM e o tswTCM, os pacotes que respeitam a taxa contratada são marcados a verde (os pacotes entre as duas taxas são marcados a amarelo e os restantes pacotes são marcadas a vermelho). As taxas dos pacotes marcados a verde e a amarelo nos dois mecanismos são superiores às taxas contratada e máxima aceitável, respectivamente. No caso do trTCM, as taxas são ligeiramente superiores devido à rajada máxima de pacotes de cada taxa aceite. O segundo mecanismo não é tão rígido como o primeiro na definição dos limiares para a troca da cor na marcação dos pacotes. No caso do tswTCM, o seu método iterativo permite marcar mais pacotes a verde e a amarelo do que no caso do trTCM.

A escolha sobre qual o mecanismo a escolher depende principalmente do que se pretende. Se a escolha for controlar uma determinada taxa e a rajada máxima de pacotes aceite, então o método mais simples, o *Token bucket*, é o escolhido. Se for necessário controlar uma determinada taxa e dois limiares do tamanho da rajada de pacotes aceite, o escolhido é o srTCM. Para controlar 2 determinadas taxas e a rajada máxima de pacotes de cada taxa aceite, o escolhido é o trTCM. Finalmente, para controlar 2 determinadas taxas, o mecanismo escolhido é o tswTCM.

8. QoS – Qualidade de Serviço

A evolução das redes de comunicações possibilitou o surgimento de novos serviços e aplicações, tais como videoconferência, VoIP (voz sobre IP) ou *Video on Demand* (figura 8.1). Estas novas aplicações necessitam de garantias de qualidade de serviço por parte da rede, como por exemplo valores reduzidos do rácio de pacotes perdidos, ou do atraso médio dos pacotes na rede, ou larguras de banda mínimas, de modo a funcionarem com a qualidade necessária. Por exemplo, não é tolerável ser recebida a seguinte mensagem numa chamada telefónica, “Não t cons go per eber, a tu voz v m aos c rte”; ou numa videoconferência obter uma imagem que não esteja nítida e em que a voz não esteja sincronizada com a imagem; ou receber neste momento dados que eram necessários para tomar uma decisão à 2 minutos atrás.

O IETF (*Internet Engineering Task Force*) formou grupos de trabalho para definir a implementação da QoS nas redes IP. Todo o tipo de rede, independentemente de se tratar de uma pequena rede de uma empresa, de um ISP (*Internet service provider*) ou de uma rede mais complexa de uma grande empresa pode tirar proveito da implementação da QoS. A QoS é definida como sendo um conjunto de características e funcionalidades que dinamicamente controlam e satisfazem requisitos de aplicações e serviços sensíveis a descartes, atrasos e variações dos atrasos (*jitter*) dos pacotes. A implementação da QoS permite que as aplicações consigam uma eficiente utilização dos recursos existentes. Os requisitos de rede dependem das aplicações usadas.

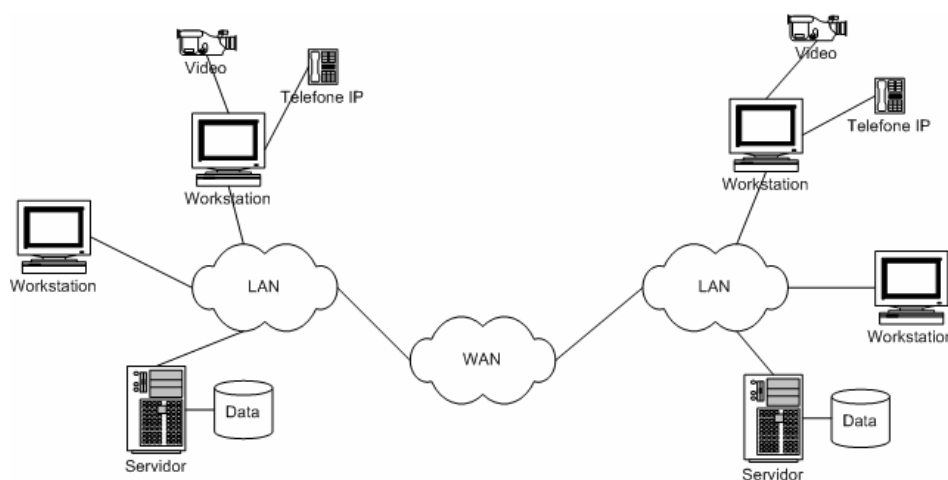


Figura 8.1 – Exemplo de uma rede multi-serviços.

O atraso de um pacote é definido como sendo o tempo necessário para atingir o destino após ter sido enviado pela origem. Esse atraso, designado por atraso ponto-a-ponto, é composto por uma componente fixa e uma variável. Na componente fixa são contabilizados dois atrasos: o atraso de propagação, que é o tempo que os *bits* de um pacote demoram a atravessar o meio físico (cabo ou ar), e o atraso de processamento, que é o tempo que o equipamento demora a processar o pacote desde que o recebe na sua interface de entrada até que o coloca na fila de espera da sua interface de saída. Na componente variável é contabilizado o atraso que depende do nível de congestionamento da rede, que é o tempo que o pacote tem de permanecer nas filas de espera das interfaces de saída dos comutadores até ser transmitido. A variação do atraso, também designada por *jitter*, é a diferença do atraso ponto-a-ponto entre os pacotes. O controlo dessa variação é importante nas transmissões de voz e vídeo, onde é preferível que todos os pacotes sofram um maior atraso a que haja uma grande variação desses atrasos. O descarte de pacotes é normalmente expresso em percentagem e representa o número de pacotes transmitidos pelo emissor que não chegam ao receptor. O descarte de pacotes normalmente ocorre quando existe congestionamento na rede e a capacidade das filas de espera é excedida.

Os equipamentos de comutação das redes, tais como os *switches* e os *routers*, são cada vez mais sofisticados e com capacidades cada vez mais elevadas de comutação. Mas por muita elevadas que sejam as capacidades de comutação, se por exemplo existirem diferentes capacidades nas ligações a montante e a jusante ou agregação de fluxos de tráfego num determinado ponto da rede (figura 8.2), poderá existir congestionamento de pacotes nesses equipamentos. Nesse momento se não existirem mecanismo próprios para combater esse congestionamento, serão descartados pacotes e os requisitos de qualidade poderão não ser cumpridos.

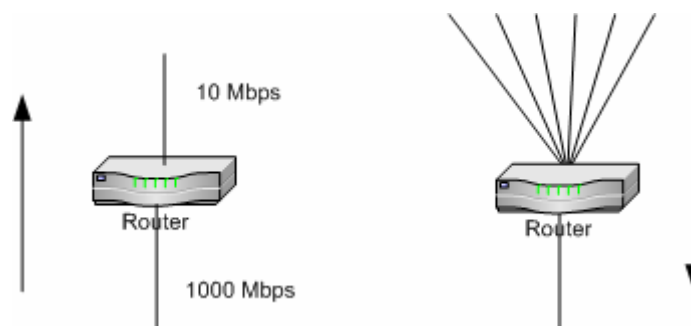


Figura 8.2 – Situações com necessidade de QoS

A administração da QoS de uma rede é complicada, porque muitas aplicações geram tipos de tráfegos imprevisíveis. Por exemplo, é difícil antecipar os padrões de uso de aplicações como o *web browsing*, o correio electrónico e aplicações de transferência de ficheiros. Neste complexo e imprevisível cenário, o administrador da rede tem que garantir a QoS de certas aplicações fulcrais na sua organização, mesmo durante os picos de tráfego na rede. Os mecanismos da QoS garantem que, se um certo tipo de fluxo de tráfego é o mais importante para uma empresa, então esse fluxo usufruirá de um uso prioritário e eficiente dos recursos da rede.

Este capítulo é organizado da seguinte forma. A secção 8.1 descreve as arquitecturas de QoS propostas pelo IETF. O módulo DiffServ presente no NS é descrito na secção 8.2. A secção 8.3 apresenta as simulações efectuadas neste capítulo. As conclusões deste capítulo são descritas na secção 8.4.

8.1 Arquitecturas QoS

Na Internet tradicional os pacotes são servidos segundo um serviço designado por *Best-effort*. Neste tipo de serviço, os pacotes são transmitidos assim que possível, havendo apenas a garantia de que a rede fará o melhor possível para os entregar no destino correcto. As filas de espera são do tipo FIFO e não é portanto possível distinguir os fluxos presentes.

As arquitecturas de QoS utilizadas nas redes IP são a *Integrated Services* (IntServ) [22] e a *Differentiated Services* (DiffServ) [33]. Nesta secção são descritas as características das arquitecturas IntServ e DiffServ e as situações em que cada uma deve ser aplicada. O serviço *Best-effort* pode ser configurado nas duas arquitecturas. As arquitecturas de QoS diferem entre si no modo como os requisitos de QoS são assegurados na passagem pela rede. Deve-se ter em atenção que a função das arquitecturas de QoS não é aumentar a largura de banda das ligações, mas sim organizar a largura de banda disponível tendo em conta o tipo de fluxo de tráfego presente.

8.1.1 *Integrated Services* - IntServ

Nesta arquitectura, também designada de *hard* QoS, é necessário cativar (reservar) recursos da rede em cada nó para cada fluxo, desde o emissor até ao receptor. Essa reserva é feita através de um protocolo designado por *Resource Reservation Protocol* (RSVP), que é activado em todos os nós da rede. O emissor antes de transmitir dados envia um pedido de transmissão via RSVP,

no qual informa a rede do seu perfil de tráfego e dos requisitos pretendidos (largura de banda e atraso). O emissor só transmite quando recebe a confirmação que os requisitos de QoS são aceites e configurados na rede. O IETF definiu duas classes de serviço nesta arquitectura, a *Guaranteed Service* [24] e a *Controlled load Service* [23].

O protocolo RSVP efectua controlo de admissão baseado na informação dos pedidos das aplicações e nos recursos de rede disponíveis. O protocolo pode aceitar ou rejeitar um pedido de reserva de largura de banda. O RSVP garante a QoS requerida pela aplicação, enquanto ela respeitar o perfil de tráfego enunciado no seu pedido. O tipo de tráfego e os requisitos de qualidade são especificados na aplicação. O fluxo é analisado antes de entrar na rede, sendo descartado se não respeitar o especificado. Deste modo os *routers* podem reservar recursos (larguras de bandas, espaço em filas de espera, etc...) e adaptar o escalonamento de pacotes garantindo a QoS requerida pelo fluxo ponto-a-ponto. O fluxo é descrito segundo os parâmetros de um *Leaky Bucket*: a taxa de chegada do *Token* (r) e a profundidade do *Bucket* (b).

O modelo *Guaranteed Service* garante a largura de banda requerida, a ausência de perdas nas filas de espera e impõe limites rígidos ao atraso ponto-a-ponto na rede, desde que as características do fluxo estejam dentro do perfil de tráfego enunciado no seu pedido. Este modelo é invocado quando a aplicação especifica os valores de TSpec (*Traffic descriptor*) e RSpec (*Service specification*). A variável TSpec descreve a fonte de tráfego com os seguintes parâmetros: a taxa do *Token bucket* (r) em *bytes/s*, a taxa de pico do fluxo (p) em *bytes/s*, a profundidade do *Bucket* (b) em *bytes*, a unidade mínima a ser policiada (m) em *bytes* (os pacotes de tamanho inferior são considerados deste tamanho para efeito de policiamento) e tamanho máximo do pacote (M) em *bytes*. A variável RSpec descreve através de dois parâmetros os requisitos de serviço: a taxa de serviço (R) em *bytes/s*, que dá a indicação da largura de banda requerida; e um valor de folga (S) em micro segundos, que representa a quantidade de atraso que um nó pode adicionar de modo a que o valor do atraso ponto-a-ponto permaneça dentro do especificado nos requisitos. O fluxo de tráfego tem que ser policiado nos nós de acesso para garantir que seja respeitado o especificado na variável TSpec, enquanto que nos nós interiores apenas é efectuado uma reformatação do fluxo. Dados os valores de TSpec e RSpec é possível calcular o maior atraso ponto-a-ponto (pior cenário) de um determinado fluxo e o espaço nas filas de espera dos *routers* que é necessário reservar. O facto de existirem reservas de largura de banda para um determinado fluxo não significa que esse recurso seja desperdiçado se o fluxo não o utilizar, podendo ser temporariamente utilizado pelos outros fluxos. Este serviço é apropriado

para aplicações que exigem garantias rígidas da limitação do atraso e a inexistência de perdas em filas de espera.

Ao contrário do modelo anterior, o modelo *Controlled Load Service* não garante qualquer largura de banda, nem impõe limites no atraso ponto-a-ponto. O valor *TSpec* é utilizado para especificar no *router* que se pretende utilizar o modelo *Controlled Load Service*, tal como acontece no modelo *Guaranteed Service*, embora não seja necessário definir a taxa de pico do fluxo. Se o fluxo é aceite, então o *router* compromete-se em oferecer ao fluxo um serviço equivalente ao *Best-effort* com a rede submetida a baixos níveis de carga. A diferença entre o *Best-effort* e o *Controlled Load Service* é que com o segundo modelo o fluxo não sentirá o aumento da carga na rede. Assim, o fluxo sofrerá uma percentagem de pacotes perdidos pouco superior à introduzida pelo próprio meio físico e um atraso total pouco superior ao mínimo possível na rede. Este serviço é apropriado para aplicações multimédia adaptativas de tempo real, tal como a videoconferência. No entanto, não é adequado para aplicações que exigem garantias rígidas quanto às perdas e aos atrasos introduzidos na rede.

A arquitectura IntServ permite configurar, através da reserva de recursos utilizando o RSVP, a QoS por fluxo. As garantias oferecidas podem ser rígidas quando utilizado o modelo *Guaranteed Service* ou puramente estatísticas utilizando o modelo *Controlled Load Service*. Todos os *routers* têm que suportar o protocolo RSVP para que a QoS seja efectiva e como as garantias são atribuídas por fluxo isso obriga os *routers* a manter a informação de cada fluxo. Este facto limita a implementação do modelo IntServ em larga escala, em cenários onde existem um elevado número de fluxos em simultâneo. O aparecimento deste problema tornou necessário a concepção de um novo modelo para garantir a QoS nas redes, surgindo a arquitectura DiffServ.

8.1.2 Differentiated Services - DiffServ

Esta arquitectura é também designada de *soft* QoS. Nesta arquitectura os fluxos de tráfego com os mesmos requisitos de rede são agregados em classes e algumas classes de tráfego recebem um melhor tratamento que as restantes. A QoS não é analisada fluxo a fluxo, mas sim por classes de tráfego. Ao contrário da arquitectura IntServ não é necessário solicitar à rede se existem condições para oferecer uma determinada QoS antes de transmitir os dados. A rede tenta oferecer em cada salto uma determinada QoS baseada na informação do campo DSCP (*Differentiated Services Code Points*) do cabeçalho IP de cada pacote. Desta forma os pacotes dos

fluxos com os mesmos requisitos são agregados no interior da rede e tratados do mesmo modo e não é necessário guardar nos *routers* a informação da QoS requerida por cada pacote. No caso do IPv4 o campo DSCP é representado nos 6 *bits* mais significativos do *byte* ToS e no caso do IPv6 no *byte Traffic Class*.

O conceito da análise da QoS em cada salto é designado por PHB (*Per-Hop Behavior*). Os nós que suportam DiffServ utilizam os DSCP para escolher para cada pacote qual o procedimento a efectuar, ou seja qual o PHB. Os PHBs são implementados baseados nos mecanismos presentes nas filas de espera, tais como os algoritmos de escalonamento e as técnicas de descarte. No entanto, a especificação do PHB é feita com base em características relevantes de um determinado serviço e não nos mecanismos usados. É possível utilizando diferentes mecanismos implementar o mesmo PHB. Dois grupos de PHB foram já normalizados pelo IETF, o *Assured Forwarding* (AF) [25] [26] e o *Expedited Forwarding* (EF) [27] [28].

O grupo AF de PHBs oferece um conjunto de quatro classes de serviço. Dentro de cada uma das classes, existem três níveis de *Drop Precedences* (DP): baixa, média e elevada probabilidade de descarte de pacotes. A tabela 8.1 apresenta os DSCP recomendados para cada um dos AF. Este grupo de PHBs garante que pacotes pertencentes à mesma classe AF não são reordenados, mesmo que tenham níveis de DP distintos.

	Classe 1	Classe 2	Classe 3	Classe 4
Baixa DP	001010	010010	011010	100010
	AF11	AF21	AF31	AF41
	DSCP 10	DSCP 18	DSCP 26	DSCP 34
Média DP	001100	010100	011100	100100
	AF12	AF22	AF32	AF42
	DSCP 12	DSCP 20	DSCP 28	DSCP 36
Elevada DP	001110	010110	011110	100110
	AF13	AF23	AF33	AF43
	DSCP 14	DSCP 22	DSCP 30	DSCP 38

Tabela 8.1 – DSCP recomendados para *Assured Forwarding* (AF)

Nos nós que suportam AF, os períodos curtos de congestionamentos são controlados utilizando a capacidade das filas de espera presentes. Em períodos mais alargados, o congestionamento deve ser controlado através do descarte de pacotes. Estes descartes não devem no entanto acontecer apenas quando se esgota a capacidade da fila de espera, mas sim gradualmente, utilizando uma variante da técnica de descarte RED. A implementação do RED sofre algumas alterações de modo a poder ser configurável por classe e por DP, surgindo por

exemplo a variante GRED (*generalized RED*). As técnicas de descarte são descritas no capítulo 5.

O PHB EF (*expedited forwarding*) tem como objectivo oferecer baixos valores de atraso, baixos valores de *jitter*, perdas de pacotes reduzidas e um serviço de largura de banda garantida (como o de uma linha dedicada virtual). Isto implica que na passagem por cada nó, os pacotes têm de encontrar as filas de espera vazias ou quase vazias, ou seja, a taxa de serviço tem que ser superior à taxa de chegada. O PHB EF garante que em cada nó a taxa de serviço é fixa e independente da intensidade de tráfego dos outros fluxos de tráfegos presentes. Em nós particulares da rede, os nós fronteira, o fluxo de tráfego é formatado e/ou policiado de modo a que a taxa de chegada não seja superior à taxa de serviço de nenhum nó. Em termos de implementação, os nós que suportam o PHB EF devem possuir por exemplo mecanismos de escalonamento com prioridades. No entanto, para que os fluxos deste tipo não influenciem negativamente os outros fluxos presentes, deve existir limite superior para os seus débitos, eliminando os pacotes que excedam esse limite. O valor do DSCP recomendado para o PHB EF é o 101110.

A arquitectura DiffServ utiliza mecanismos de classificação e condicionamento de tráfego na fronteira da rede, e implementa PHBs ao longo do caminho. Dois conceitos importantes na arquitectura DiffServ são os de SLA (*Service Level Agreement*) e TCA (*Traffic Conditioning Agreement*). O SLA é um contrato celebrado entre o cliente e o fornecedor do serviço que contém a especificação do serviço a ser prestado. No contrato TCA são especificadas regras de classificação, medição, marcação, policiamento, eliminação e/ou formatação de pacotes.

Na arquitectura DiffServ é feita uma clara distinção entre nó fronteira (*Edge node*) e nó interior (*Core node*) de um domínio DS. Um domínio DS é caracterizado por um conjunto de nós interligados, que suportam a arquitectura DiffServ, nos quais é comum um conjunto de PHBs implementados e a política de fornecimento de serviço. Os nós fronteira, como o próprio nome indica, são responsáveis pela ligação entre os diversos domínios, que suportam ou não DiffServ. O nó fronteira controla o fluxo de tráfego que entra na rede de modo a garantir que ele seja conforme o TCA especificado entre o domínio DS e o domínio exterior. Os nós fronteira também são responsáveis pela marcação/remarcação dos pacotes. Os nós internos requerem um número reduzido de funcionalidades, usam o DSCP para classificação e os

mecanismos de presentes nas filas de espera (técnicas de descarte e algoritmos de escalonamentos) essenciais para implementação dos PHBs.

Os nós fronteira garantem a conformidade dos fluxos de tráfego com os TCA presentes, utilizando as seguintes funcionalidades: classificação, marcação, medição, policiamento e *shaping* (formatação). A classificação consiste na selecção dos pacotes baseada, ou em múltiplos campos do cabeçalho IP (*Multi-Field Classification* – MF), como por exemplo, endereço de origem e destino, portas, protocolos, etc, ou no campo DSCP (*Behavior Aggregate Classification* – BA). A marcação marca o pacote, colocando no campo DS-Field o DSCP específico; essa funcionalidade marca cada pacote com o PHB a aplicar. A medição mede as propriedades temporais do tráfego previamente classificado. Após comparação com o perfil de tráfego determinado pelo TCA, essa informação é passada aos restantes elementos para que tomem as acções apropriadas para regular o fluxo de tráfego conforme o pacote esteja *in-profile* ou *out-of-profile*. O policiamento garante que o fluxo de tráfego está em conformidade com o perfil especificado, eliminando ou remarcando para prioridades mais baixas o tráfego *out-of-profile*. Os mecanismos de policiamento são por exemplo, srTCM [29] (*Single Rate Three Color Marker*), trTCM [30] (*Two Rate Three Color Marker*) e tswTCM [31] (*Time Sliding Window Three Colour Marker*). Estes mecanismos são descritos no capítulo 7. A funcionalidade de *Shaping* regula os fluxos de tráfego, os pacotes de um determinado fluxo (ou de um conjunto de fluxo) são atrasados de modo a forçar a conformidade com o perfil adequado. O *Token bucket* e o *Leaky bucket* são mecanismos habitualmente utilizados para implementar esta funcionalidade. Estes mecanismos são descritos no capítulo 7. Cada um dos domínios DS pode suportar diferentes combinações entre DSCP e PHB. Os nós fronteira são responsáveis por mapear essas diferenças.

Os nós internos são mais simples do que os nós fronteira, possuem as seguintes funcionalidades: classificação e *queuing*. Por razões de eficiência, apenas usam o DSCP para classificação. Deste modo a utilização de apenas um campo do cabeçalho IP torna a classificação BA (*Behavior Aggregate Classification*) muito mais simples que a MF (*Multi-Field Classification*). O termo *queuing* refere-se aos mecanismos de gestão das filas de espera, que englobam os algoritmos de escalonamento e as técnicas de descarte, que são elementos essenciais na implementação dos PHBs. Os fluxos de tráfego que chegam a um nó são normalmente divididos em filas de espera separadas e tratados por algoritmos de escalonamento que decidem quais os pacotes a serem enviados de acordo com a QoS

pretendida. No capítulo 4, alguns algoritmos de escalonamento foram estudados e comparados por simulação. Variantes do FQ (*Fair Queuing*) são normalmente utilizadas para implementar o grupo de PHBs AF e o PQ (*Priority Queuing*) na implementação do grupo PHB EF. Variando de fabricante para fabricante, outros algoritmos de escalonamento são utilizados para implementar os PHBs: RR (*Round Robin*), WRR (*Weighted Round Robin*), DRR (*Deficit Round Robin*), CBQ (*Class-Based Queuing*), WFQ (*Weighted Fair Queuing*), etc... A técnica de descarte utilizada é uma variante da tradicional RED (*Random Early Detection*), descrita no capítulo 5, sendo o mecanismo utilizado para o controlo de congestionamento. As variantes existentes do RED são: WRED (*Weighted RED*), GRED (*Generalized RED*), RIO-C (*RED with In/Out and Coupled virtual Queues*) e RIO-DC (*RED with In/Out and De-Coupled Queues*).

Na secção seguinte, é analisado o módulo DiffServ presente no simulador NS, verificando quais os mecanismos presentes nos diversos componentes da arquitectura.

8.2 Implementação no NS

O módulo DiffServ do NS simula os dois grupos de PHB normalizados pelo IETF, o AF (*Assured Forwarding*) e o EF (*Expedited Forwarding*). No caso do AF é possível configurar 4 classes e 3 DP (*Drop Precedence*) por classe, o que permite diferenciação de serviço dentro da mesma classe. Os AF PHB, definidos na tabela 8.1, são mapeados directamente numa fila de espera física (que define qual a classe do pacote) e numa fila de espera virtual (que define qual a DP). As DP são simuladas utilizando para cada classe (por cada fila física), 3 filas de espera virtuais configuradas com a técnica de descarte RED, onde os parâmetros do RED definem o nível baixo, médio ou alto, da probabilidade de descarte. Cada DSCP configurado é mapeado para uma fila de espera virtual de uma determinada fila de espera física. O EF PHB é configurado utilizando uma *Priority Queuing* (PQ) como algoritmo de escalonamento, onde é possível configurar qual a taxa máxima dedicada a fila, de modo a que outras filas possam ser servidas.

Os 3 principais componentes do módulo DiffServ no NS são *Policy*, *Edge router* e *Core router*. Na *Policy* define-se qual a política de serviço, ou seja, qual o nível de serviço que uma determinada classe deverá receber dentro da rede. O *Edge router* (nó fronteira) marca os pacotes com os DSCP especificados na *Policy*. O *Core router* (nó interior) analisa o DSCP de cada pacote e encaminha os pacotes de acordo com o marcado.

Os mecanismos descritos na secção anterior são na sua maioria possíveis de configurar no NS. No nó fronteira é possível configurar para além dos 3 mecanismos de policiamento enunciados anteriormente (srTCM, trTCM e tswTCM), mais 2, o tsw2CM (*Time Sliding Window with 2 Color Marking*) e o *Token bucket*. Cada nó fronteira possui funções específicas para classificação, marcação, medição e policiamento. Todos os fluxos de tráfego entre o mesmo nó destino e nó origem serão tratados como um único fluxo agregado, ou seja a classificação MF é apenas baseada no endereço do nó origem e no endereço do nó destino. As técnicas de descarte presentes neste módulo do NS são a RIO-C, RIO-DC, WRED e DROP. A última é similar à técnica DropTail com o limite da fila de espera igual ao limiar inferior da técnica RED, sendo os pacotes todos descartados quando esse valor é ultrapassado. Os algoritmos de escalonamento possíveis de configurar são o RR (*Round Robin*), WRR (*Weighted Round Robin*), PRI (*Priority Queuing*) e WIRR (*Weighted Interleaving Round Robin*).

8.3 Simulações

O objectivo desta secção é estudar por simulação o efeito da implementação da QoS num determinado cenário, utilizando a arquitectura DiffServ. O cenário escolhido é apresentado na figura 8.3.

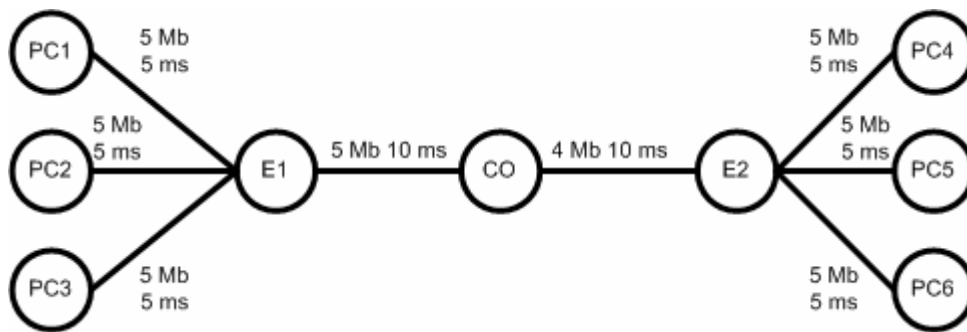


Figura 8.3 – Cenário da Simulação para o estudo da QoS

Cada simulação tem a duração de 15 segundos. Todas as filas de espera são configuradas com capacidade de 50 pacotes. Os nós E1 e E2 representam os nós fronteira e o nó CO representa um nó interior. A ligação entre E1 e CO tem capacidade de 5 *Mbits/s*, a ligação entre CO e E2 tem capacidade de 4 *Mbits/s* e ambas possuem um atraso de propagação de 10 ms. Os nós PC1 a PC6 representam clientes ligados à rede. São estabelecidas três sessões de tráfego com as mesmas características entre PC1 e PC4 (fluxo 1), PC2 e PC5 (fluxo 2) e PC3 e PC6 (fluxo 3). O protocolo de transporte escolhido é o UDP e as fontes de tráfego geram pacotes a uma

taxa de 4 *Mbits/s*, com intervalo entre chegadas exponencialmente distribuído, com pacotes de tamanho fixo e igual a 1000 *bytes*. Os clientes são ligados aos nós fronteira com ligações de 5 *Mbits/s* e com atrasos de propagação de 5 ms. Todas as sessões são iniciadas em simultâneo no início da simulação e durante 15 segundos, o que provoca uma situação de congestionamento na rede.

As subsecções seguintes apresentam os resultados obtidos no caso de existir ou não QoS. As simulações efectuadas estão divididas em 3 secções: a primeira secção apresenta os resultados obtidos quando o cenário escolhido é simulado sem QoS, ou seja os três fluxos são tratados de igual modo e servidos assim que possível (*Best-effort*). A segunda secção apresenta os resultados obtidos quando a rede é configurada com QoS, onde num dos fluxos de tráfego é implementado o PHB AF e os restantes são servidos em *Best-effort*. A última secção apresenta os resultados obtidos quando é configurado o PHB EF num dos fluxos de tráfego e os restantes são servidos em *Best-effort*.

8.3.1 Serviço *Best-effort*

Esta secção tem por objectivo apresentar os resultados obtidos quando o cenário da figura 8.4 é simulado sem a presença de mecanismos de QoS. Todas as filas de espera são configuradas com um algoritmo de escalonamento FIFO e com DropTail como técnica de descarte, mecanismos estudados nos capítulos 4 e 5, respectivamente. Teoricamente os 3 fluxos vão partilhar equitativamente os 5 *Mbits/s* da primeira ligação e os 4 *Mbits/s* da segunda ligação, sofrendo uma percentagem de descarte apresentada nas tabelas 8.2 e 8.3, na primeira e segunda ligação partilhada, respectivamente.

Fluxo	Taxa transmissão (<i>Mbits/s</i>)	Largura de banda (<i>Mbits/s</i>)	Percentagem descartes (%)
1	4	1.66	60
2	4	1.66	60
3	4	1.66	60

Tabela 8.2 – Valores teóricos para a ligação E1->CO com *Best-effort*

Fluxo	Taxa transmissão (Mbits/s)	Largura de banda (Mbits/s)	Percentagem descartes (%)
1	1.66	1.33	18.75
2	1.66	1.33	18.75
3	1.66	1.33	18.75

Tabela 8.3 – Valores teóricos para a ligação CO->E2 com *Best-effort*

O cenário é simulado 10 vezes e as ligações partilhadas pelos fluxos são analisadas e os resultados obtidos são apresentados nas tabelas 8.4 e 8.5, com intervalos de confiança de 90%, para as ligações E1-CO e CO-E2, respectivamente.

Fluxo	Taxa transmissão (Mbits/s)	Largura de banda (Mbits/s)	Percentagem descartes (%)
1	[3.97, 4.04]	[1.55, 1.69]	[57.99, 61.16]
2	[3.97, 4.03]	[1.66, 1.77]	[55.72, 58.46]
3	[3.96, 4.02]	[1.63, 1.73]	[56.85, 58.90]

Tabela 8.4 – Resultados obtidos para a ligação E1->CO com *Best-effort*

Fluxo	Taxa transmissão (Mbits/s)	Largura de banda (Mbits/s)	Percentagem descartes (%)
1	[1.55, 1.69]	[1.24, 1.36]	[19.51, 19.91]
2	[1.66, 1.77]	[1.34, 1.42]	[19.31, 19.64]
3	[1.63, 1.73]	[1.32, 1.40]	[19.11, 19.49]

Tabela 8.5 – Resultados obtidos para a ligação CO->E2 com *Best-effort*

Os resultados obtidos e apresentados nas tabelas anteriores são complementados com uma análise gráfica da evolução temporal da largura de banda utilizada em cada ligação, respectivamente nas figuras 8.4 e 8.5.

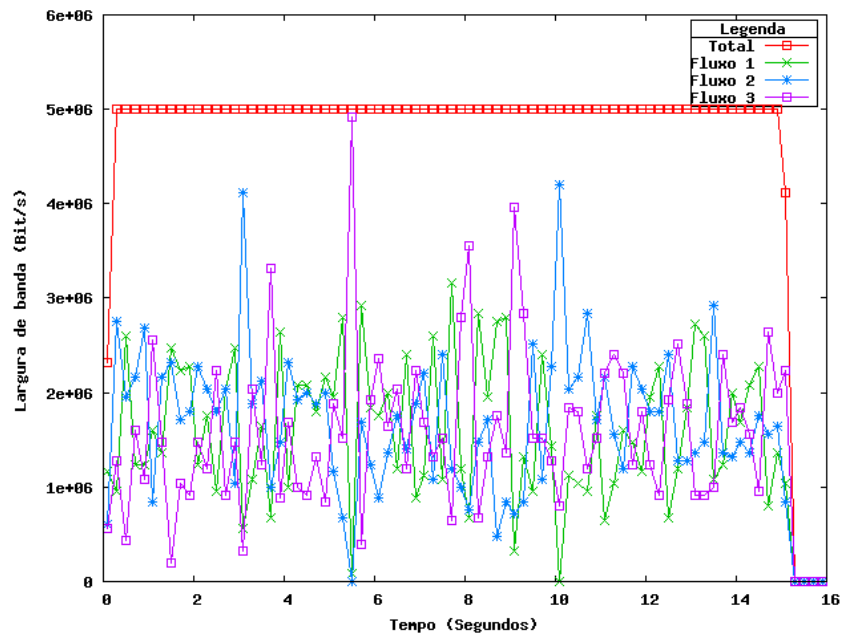


Figura 8.4 – Largura de banda utilizada por cada fluxo na ligação E1->CO com *Best-effort*

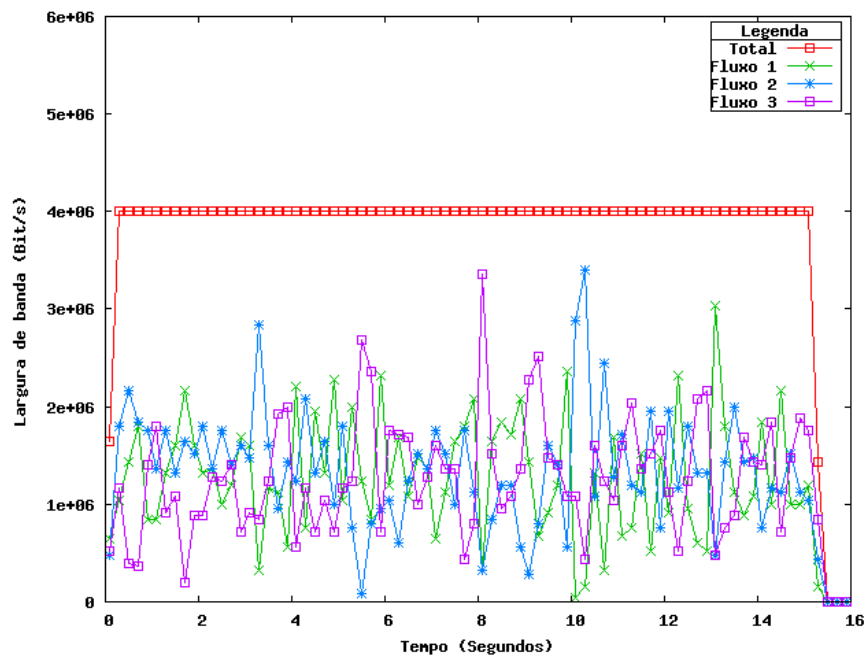


Figura 8.5 – Largura de banda utilizada por cada fluxo na ligação CO->E2 com *Best-effort*

Analisando os resultados obtidos na tabela 8.4 e na tabela 8.5, verifica-se que os 3 fluxos partilham equitativamente as ligações partilhadas, de acordo com o esperado teoricamente e apresentado nas tabelas 8.2 e 8.3. A percentagem de descarte dos três fluxos na primeira ligação partilhada é, como seria de esperar, muito elevada, cerca de 60 % dos pacotes enviados pelos PC são descartados. A redução de 1 *Mbits/s* da capacidade da segunda ligação partilhada provoca, durante os 15 segundos de transmissão, o descarte de cerca de 19 % dos pacotes que chegam à ligação (tabela 8.3), valor concordante com o da tabela 8.5. Os 3 fluxos são tratados do mesmo modo, não existindo distinção entre eles. As figuras 8.4 e 8.5 ilustram a partilha equitativa da largura de banda, mas também o uso completamente aleatório da largura de banda, um fluxo pode num determinado momento transmitir a mais de 3 *Mbits/s* e no instante seguinte a menos de 1 *Mbits/s*. Num cenário *Best-effort* não existe a possibilidade de garantir por exemplo uma determinada largura de banda para um determinado fluxo de tráfego.

8.3.2 Serviço DiffServ – PHB AF

Esta secção tem por objectivo apresentar os resultados obtidos, quando se configura no cenário da figura 8.4 os mecanismos de QoS com PHB AF.

Para efeito de simulação é considerado como vital para o negócio o fluxo 3, configurando-o com um PHB AF e reservando uma largura de banda de 3 *Mbits/s*. Os outros fluxos de tráfego são considerados de igual importância e partilham entre si os restantes recursos (*Best-*

effort), 2 *Mbits/s* na primeira ligação e 1 *Mbits/s* na segunda ligação partilhada. No domínio DS é configurada apenas uma classe (uma fila de espera física em cada nó) com dois níveis de descarte (duas filas de espera virtuais). O algoritmo de escalonamento escolhido é o RR e a técnica de descarte presente é a RIO-C. A técnica de descarte é configurada do seguinte modo: no caso do nível de descarte elevado com limiar inferior igual a 10 pacotes e superior igual a 20 pacotes com uma probabilidade de descarte de 0,1; no caso do nível de descarte baixo com limiar inferior igual a 20 pacotes e superior igual a 40 pacotes com uma probabilidade de descarte de 0,02. O fluxo de tráfego vital é policiado com um mecanismo *Token bucket* configurado para um débito de 3*Mbit/s* e tamanho do *burst* igual a 5000 *bytes*. O CP (*Code point*) do fluxo vital é igual a 10 e todo o restante fluxo de tráfego tem um CP igual a 12. Os pacotes com CP igual a 10 são encaminhados para a fila virtual com menor nível de descarte e os com CP igual a 12 para a outra fila virtual. Os valores teóricos são apresentados nas tabelas 8.6 e 8.7, para cada fluxo e para cada CP.

	Taxa de transmissão (<i>Mbits/s</i>)	Largura de banda (<i>Mbits/s</i>)	Percentagem descartes (%)
Fluxo 1	4	1	75
Fluxo 2	4	1	75
Fluxo 3	4	3	25
CP 10	3	3	0
CP 12	9	2	77.77

Tabela 8.6 – Valores teóricos para a ligação E1->CO com DiffServ - PHB AF

	Taxa transmissão (<i>Mbits/s</i>)	Largura de banda (<i>Mbits/s</i>)	Percentagem descartes (%)
Fluxo 1	1	0.5	50
Fluxo 2	1	0.5	50
Fluxo 3	3	3	0
CP 10	3	3	0
CP 12	2	1	50

Tabela 8.7 – Valores teóricos para a ligação CO->E2 com DiffServ - PHB AF

O cenário é simulado 10 vezes e as ligações partilhadas pelos fluxos são analisadas e os resultados obtidos são apresentados nas tabelas 8.8 e 8.9, com intervalos de confiança de 90%, para as ligações E1-CO e CO-E2, respectivamente.

	Taxa de transmissão (Mbits/s)	Largura de banda (Mbits/s)	Percentagem descartes (%)
Fluxo 1	[3.96, 4.00]	[0.93, 0.95]	[76.09, 76.75]
Fluxo 2	[3.94, 4.00]	[0.92, 0.94]	[76.33, 76.86]
Fluxo 3	[3.97, 4.02]	[3.13, 3.15]	[20.76, 21.79]
CP 10	[2.97, 2.98]	[2.93, 2.94]	[0.99, 1.30]
CP 12	[8.92, 9.04]	[2.07, 2.08]	[76.75, 77.06]

Tabela 8.8 – Resultados obtidos para a ligação E1->CO com DiffServ - PHB AF

	Taxa de transmissão (Mbits/s)	Largura de banda (Mbits/s)	Percentagem descartes (%)
Fluxo 1	[0.93, 0.95]	[0.48, 0.50]	[47.04, 48.34]
Fluxo 2	[0.92, 0.94]	[0.48, 0.51]	[46.30, 47.67]
Fluxo 3	[3.13, 3.15]	[3.02, 3.04]	[3.36, 3.59]
CP 10	[2.93, 2.94]	[2.92, 2.93]	[0.34, 0.40]
CP 12	[2.07, 2.08]	[1.09, 1.10]	[47.30, 47.52]

Tabela 8.9 – Resultados obtidos para a ligação CO->E2 com DiffServ - PHB AF

Nas tabelas anteriores, o valor referente a percentagem de descartes é igual à soma de 2 parcelas: a primeira refere-se aos descartes provocados devido a falta de recursos nas filas de espera; e a segunda refere-se ao número de pacotes descartados pelo mecanismo RED de modo a evitar a falta de recursos nas filas de espera.

Os resultados obtidos e apresentados nas tabelas anteriores são complementados com uma análise gráfica da evolução temporal da largura de banda utilizada em cada ligação, respectivamente nas figuras 8.6 e 8.7.

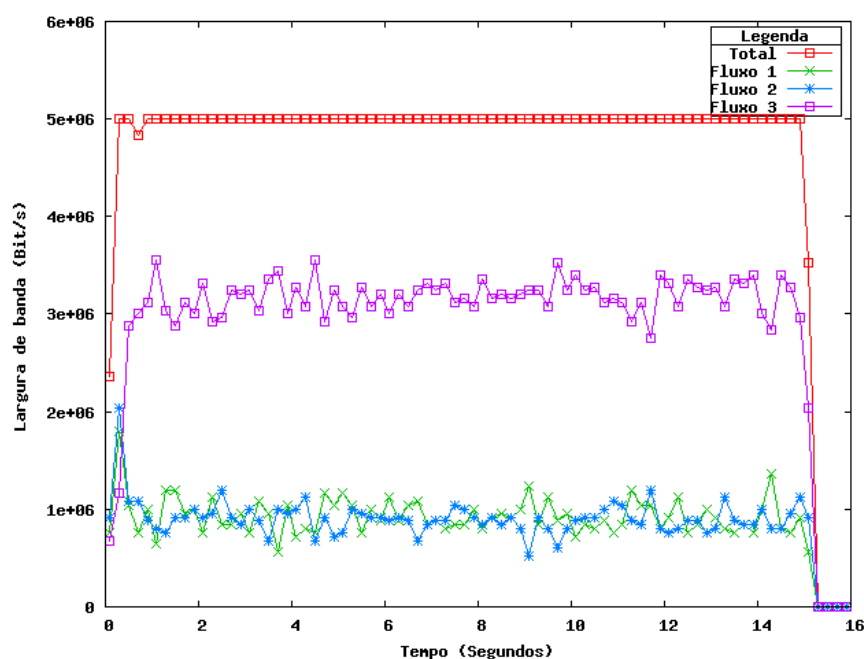


Figura 8.6 – Largura de banda utilizada na ligação E1->CO com DiffServ - PHB AF

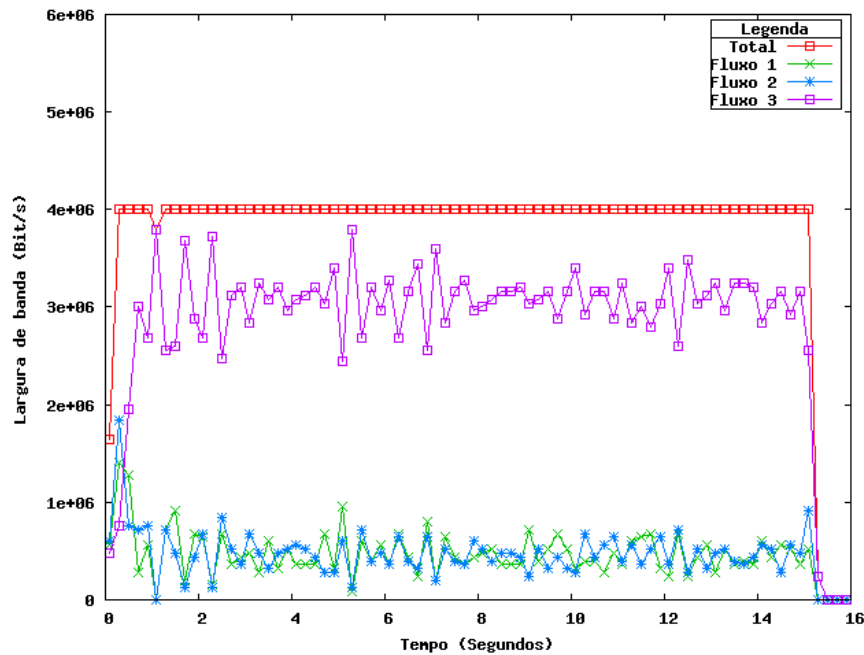


Figura 8.7 – Largura de banda utilizada na ligação CO->E2 com DiffServ - PHB AF

Os valores obtidos por simulação são coerentes com os calculados teoricamente. O policiamento é um mecanismo que é possível visualizar quando analisados os resultados obtidos na tabela 8.8 para o fluxo 3 e o CP 10: dos 4 *Mbits/s* do fluxo 3 apenas 3 *Mbits/s* foram classificados com CP igual a 10, como seria de esperar. Os pacotes *out-of-profile* da classe são classificados com CP igual a 12 e são tratados como os pacotes dos fluxos 1 e 2, daí a largura de banda do fluxo 3 ser ligeiramente superior à do CP 10. Analisando a figura 8.6 verifica-se também que a taxa de transmissão do fluxo 3 é ligeiramente superior a 3 *Mbits/s*, ou seja alguns pacotes do fluxo classificados como *out-of-profile* também são enviados. Na figura 8.6 também é possível verificar que a taxa de transmissão de cada um dos 3 fluxos sofre menores oscilações, comparando com a situação de *Best-effort* da figura 8.4, porque existe uma formatação dos fluxos que entram no domínio DS. O fluxo com CP igual a 10, fluxo considerado fulcral, praticamente não sente a redução de 1 *Mbits/s* da capacidade da segunda ligação partilhada: as figuras 8.6 e 8.7 e as tabelas 8.8 e 8.9 ilustram o uso de aproximadamente 3 *Mbits/s* da largura de banda de ambas as ligações.

Consideremos agora um outro cenário, em que para o fluxo de tráfego vital apenas é necessário garantir um débito de 2 *Mbits/s*, o mecanismo de policiamento é reconfigurado para esse valor. Os pacotes do fluxo 3 que excedam este débito são tratados como os pacotes dos outros tráfegos, ou seja, são marcados com CP igual a 12. Os valores teóricos são apresentados nas tabelas 8.10 e 8.11.

	Taxa de transmissão (<i>Mbits/s</i>)	Largura de banda (<i>Mbits/s</i>)	Percentagem descartes (%)
Fluxo 1	4	1.5	62.5
Fluxo 2	4	1.5	62.5
Fluxo 3	4	2	50
CP 10	2	2	0
CP 12	10	3	70

Tabela 8.10 – Valores teóricos para a ligação E1->CO com DiffServ - PHB AF, após a reconfiguração

	Taxa transmissão (<i>Mbits/s</i>)	Largura de banda (<i>Mbits/s</i>)	Percentagem descartes (%)
Fluxo 1	1.5	1	33.33
Fluxo 2	1.5	1	33.33
Fluxo 3	2	2	0
CP 10	2	2	0
CP 12	3	2	33.33

Tabela 8.11 – Valores teóricos para a ligação CO->E2 com DiffServ - PHB AF, após a reconfiguração

Após a reconfiguração, o cenário é simulado 10 vezes e as ligações partilhadas pelos fluxos são analisadas e os resultados obtidos são apresentados nas tabelas 8.12 e 8.13, com intervalos de confiança de 90%, para as ligações E1-CO e CO-E2, respectivamente.

CP	Taxa transmissão (<i>Mbits/s</i>)	Largura de banda (<i>Mbits/s</i>)	Percentagem descartes (%)
Fluxo 1	[3.95, 4.00]	[1.21, 1.22]	[69.17, 69.52]
Fluxo 2	[3.96, 4.01]	[1.22, 1.24]	[68.80, 69.37]
Fluxo 3	[3.95, 4.00]	[2.55, 2.57]	[35.25, 36.04]
CP 10	[2.00, 2.00]	[1.97, 1.98]	[1.14, 1.43]
CP 12	[9.89, 9.98]	[3.03, 3.04]	[69.31, 69.63]

Tabela 8.12 – Resultados obtidos para a ligação E1->CO com DiffServ - PHB AF, após a reconfiguração

CP	Taxa transmissão (<i>Mbits/s</i>)	Largura de banda (<i>Mbits/s</i>)	Percentagem descartes (%)
Fluxo 1	[1.21, 1.22]	[0.82, 0.83]	[31.82, 32.43]
Fluxo 2	[1.22, 1.24]	[0.82, 0.84]	[32.45, 33.21]
Fluxo 3	[2.55, 2.57]	[2.35, 2.37]	[7.43, 7.82]
CP 10	[1.97, 1.98]	[1.97, 1.97]	[0.28, 0.42]
CP 12	[3.03, 3.04]	[2.05, 2.05]	[32.28, 32.51]

Tabela 8.13 – Resultados obtidos para a ligação CO->E2 com DiffServ - PHB AF, após a reconfiguração

Os resultados obtidos e apresentados nas tabelas anteriores são complementados com uma análise gráfica da evolução temporal da largura de banda utilizada em cada ligação, respectivamente nas figuras 8.8 e 8.9.

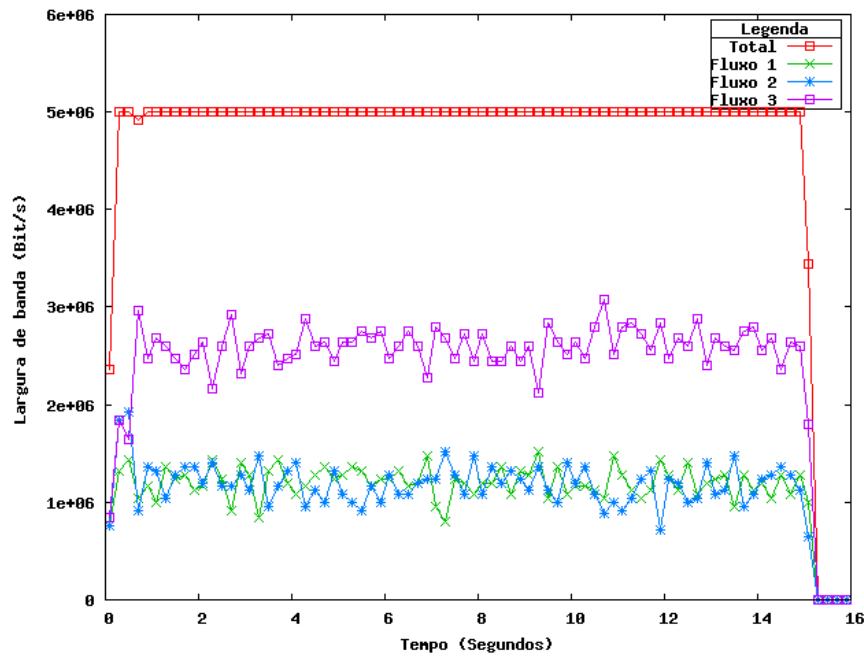


Figura 8.8 – Largura de banda utilizada na ligação E1->CO com DiffServ- PHB AF, após reconfiguração

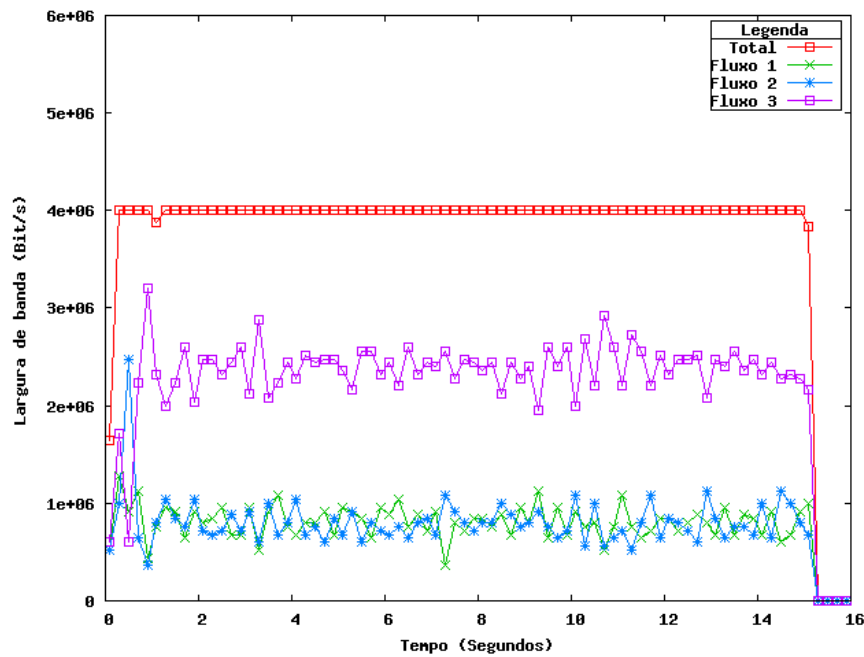


Figura 8.9 – Largura de banda utilizada na ligação CO->E2 com DiffServ- PHB AF, após reconfiguração

Após a reconfiguração do mecanismo de policiamento, apenas 2 *Mbits/s* dos 4 *Mbits/s* transmitidos pelo fluxo 3 foram classificados com CP igual a 10 (tabela 8.12). Mais uma vez alguns pacotes classificados como *out-of-profile* também são transmitidos, como se pode verificar na figura 8.8 onde a taxa de transmissão do fluxo 3 é um pouco superior ao da classe com CP igual a 10. Na segunda ligação partilhada sucede o mesmo que na situação anterior

com o mecanismo de policiamento a uma taxa de 3 *Mbits/s*: os pacotes da classe com CP igual a 10 pouco sofrem com a redução de 1 *Mbits/s* da ligação, como se pode verificar na tabela 8.13 e na figura 8.9.

O fluxo de tráfego considerado fulcral recebe por parte da rede um tratamento distinto dos restantes fluxos, mas esse tratamento não é suficiente se se pretender que não exista qualquer descarte de pacotes deste fluxo.

8.3.3 Serviço DiffServ – PHB EF

Esta secção tem por objectivo apresentar os resultados obtidos, quando se configura mecanismos de QoS com PHB EF no cenário da figura 8.4. Neste cenário considera-se que para o fluxo de tráfego vital é necessário garantir que nenhum pacote da classe seja descartado. A QoS é configurada para PHB EF e para uma taxa de 3 *Mbits/s*. Os valores teóricos esperados são apresentados nas tabelas 8.14 e 8.15.

	Taxa de transmissão (<i>Mbits/s</i>)	Largura de banda (<i>Mbits/s</i>)	Percentagem descartes (%)
Fluxo 1	4	1	75
Fluxo 2	4	1	75
Fluxo 3	4	3	25
CP 10	3	3	0
CP 12	9	2	77.77

Tabela 8.14 – Valores teóricos para a ligação E1->CO com DiffServ - PHB EF

	Taxa transmissão (<i>Mbits/s</i>)	Largura de banda (<i>Mbits/s</i>)	Percentagem descartes (%)
Fluxo 1	1	0.5	50
Fluxo 2	1	0.5	50
Fluxo 3	3	3	0
CP 10	3	3	0
CP 12	2	1	50

Tabela 8.15 – Valores teóricos para a ligação CO->E2 com DiffServ - PHB EF

Após a reconfiguração, o cenário é simulado 10 vezes e as ligações partilhadas pelos fluxos são analisadas e os resultados obtidos são apresentados nas tabelas 8.16 e 8.17, com intervalos de confiança de 90%, para as ligações E1-CO e CO-E2, respectivamente.

CP	Taxa transmissão (Mbits/s)	Largura de banda (Mbits/s)	Percentagem descartes (%)
Fluxo 1	[3.94, 4.01]	[0.97, 1.00]	[74.85, 75.68]
Fluxo 2	[3.95, 4.02]	[0.96, 0.99]	[74.98, 75.92]
Fluxo 3	[3.97, 4.01]	[3.06, 3.07]	[22.80, 23.59]
CP 10	[2.97, 2.98]	[2.97, 2.98]	0
CP 12	[8.91, 9.05]	[2.05, 2.05]	[76.97, 77.34]

Tabela 8.16 – Resultados obtidos para a ligação E1->CO com DiffServ - PHB EF

CP	Taxa transmissão (Mbits/s)	Largura de banda (Mbits/s)	Percentagem descartes (%)
Fluxo 1	[0.97, 1.00]	[0.48, 0.50]	[49.49, 50.41]
Fluxo 2	[0.96, 0.99]	[0.48, 0.50]	[49.68, 50.58]
Fluxo 3	[3.06, 3.07]	[3.02, 3.03]	[1.31, 1.54]
CP 10	[2.97, 2.98]	[2.97, 2.98]	0
CP 12	[2.05, 2.05]	[1.02, 1.03]	[50.00, 50.01]

Tabela 8.17 – Resultados obtidos para a ligação CO->E2 com DiffServ - PHB EF

Os resultados obtidos e apresentados nas tabelas anteriores são complementados com uma análise gráfica da evolução temporal da largura de banda utilizada em cada ligação, respectivamente nas figuras 8.10 e 8.11.

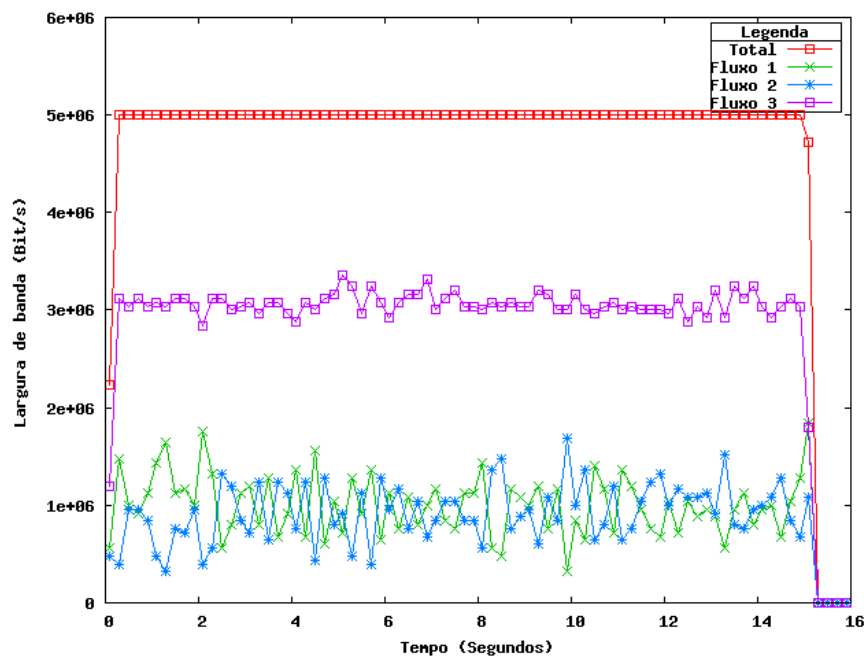


Figura 8.10 – Largura de banda utilizada na ligação E1->CO com DiffServ- PHB EF

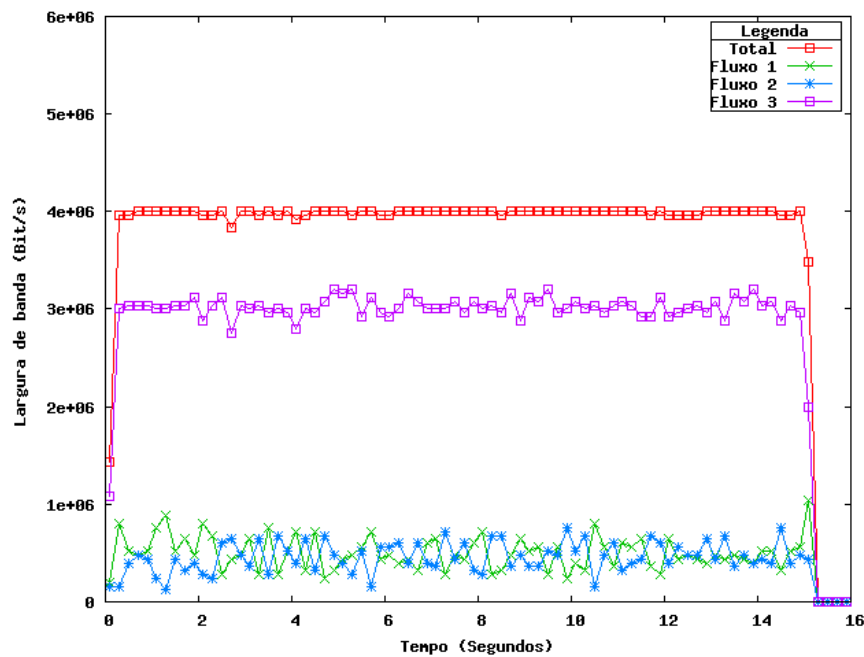


Figura 8.11 – Largura de banda utilizada na ligação CO->E2 com DiffServ- PHB EF

O mecanismo de policiamento é configurado com os mesmos parâmetros da primeira simulação com PHB AF. Na tabela 8.16 verifica-se que o número de pacotes classificados com CP igual a 10 é idêntico ao da tabela 8.8. A diferença está no número de pacotes descartados. Com PHB EF nenhum pacote da classe com CP igual a 10 é descartado, tanto na primeira como na segunda ligação partilhada como se verifica nas tabelas 8.16 e 8.17. As figuras 8.10 e 8.11 ilustram a taxa quase constante de 3 *Mbits/s* na passagem pelas ligações partilhadas, em contraste com as pequenas oscilações visualizadas na figuras 8.6 e 8.7 para o caso do PHB AF. As oscilações são provocadas pelo armazenamento dos pacotes na fila de espera: no caso do PHB AF o algoritmo de escalonamento é rotativo, ou seja, os pacotes têm que aguardar pela sua vez antes de serem transmitidos, enquanto que no PHB EF o algoritmo de escalonamento é prioritário, ou seja os pacotes são imediatamente transmitidos.

8.4 Conclusões

As grandes quantidades de informação que diariamente circulam nas redes sob a forma de voz, vídeo, imagem e dados, tornam o suporte à QoS um elemento fundamental dessas redes. O IETF formou grupos de trabalho que definiram a implementação da QoS. Destes grupos de trabalho surgiram duas arquitecturas distintas, a *Integrated Services* (IntServ) e a *Differentiated Services* (DiffServ).

A primeira arquitectura a surgir foi a IntServ, baseada num protocolo de reserva de recursos designado por RSVP. A análise da QoS é efectuada ponto-a-ponto e o serviço só é fornecido se a reserva for efectuada em todo o percurso. O IETF definiu duas classes de serviço, a *Guaranteed Service* e a *Controlled Load Service*. A primeira classe garante a largura de banda pretendida pelo fluxo e a ausência de descarte de pacotes nas filas de espera. A segunda classe não garante qualquer largura de banda, mas compromete-se em oferecer ao fluxo um serviço equivalente ao *Best-effort* numa rede submetida a baixos níveis de carga. A diferença entre o *Best-effort* e o *Controlled Load Service* é que com o segundo modelo o fluxo não sentirá o aumento da carga na rede. Esta arquitectura é orientada ao fluxo e neste sentido existem alguns problemas: a informação sobre cada fluxo tem que ser armazenada em cada nó e aumenta proporcionalmente com o número de fluxos, causando uma sobrecarga de armazenamento e processamento nos nós; todos os nós têm que suportar o protocolo RSVP para implementar a arquitectura IntServ; em comunicações de curta duração, do tipo WWW, a sobrecarga causada pela sinalização do protocolo pode deteriorar o desempenho da rede sendo percebida pela aplicação.

Na arquitectura DiffServ existe apenas um número limitado de classes de serviço, indicadas no cabeçalho IP de cada pacote (no campo DSCP). A análise da QoS é efectuada salto a salto e é designada por PHB (*Per-Hop Behavior*). Deixa de existir o conceito de fluxo e surge o conceito de agregado de fluxos, que representa um conjunto de fluxos com requisitos similares, agrupados numa mesma classe. A informação necessária a armazenar na rede deixa de ser proporcional ao número de fluxos, o que torna esta arquitectura mais escalável. Os nós da rede são classificados de nós fronteira (*Edge nodes*) ou de nós de núcleo (*Core nodes*), e são agrupados em domínios DS. Um domínio DS é um conjunto de nós interligados que suportam a arquitectura DiffServ, onde é definida uma política de fornecimento de QoS comum para todos os nós. A arquitectura DiffServ pode ser implementada mesmo em redes que suportam parcialmente a arquitectura. Os nós da rede que não suportam DiffServ simplesmente ignoram o campo DSCP dos pacotes e fornecem um serviço *Best-effort*. Dois grupos de PHB já foram normalizados pelo IETF: *Assured Forwarding* (AF) e *Expedited Forwarding* (EF). O PHB AF disponibiliza um conjunto de 4 classes cada uma com 3 níveis de probabilidade de descarte (*Drop Precedences* - DP): baixa, média e elevada. Os requisitos de cada classe são definidos nas políticas da rede. O PHB EF tem por objectivo oferecer baixos

valores de atraso, baixos valores de *jitter*, perdas de pacotes reduzidas e um serviço de largura de banda garantida (como o de uma linha dedicada virtual).

Apenas a arquitectura DiffServ foi estudada por simulação. As simulações permitiram analisar a implementação dos diversos mecanismos existentes na arquitectura DiffServ, para o PHB AF e o PHB EF. No caso do PHB AF e comparando com um serviço *Best-effort* verificou-se que um dos fluxos existente e considerado fulcral recebeu da rede um tratamento especial, ou seja uma certa QoS. A configuração da arquitectura DiffServ com um PHB EF, permitiu eliminar a percentagem de descarte sofrida pelos pacotes do fluxo fulcral com um PHB AF, ou seja nenhum pacote classificado como pertencente a classe do fluxo fulcral foi descartado. O cenário escolhido é simples, mas os princípios de configuração das diversas arquitecturas são os mesmos, como os de uma rede bem mais complexa.

A decisão sobre qual a arquitectura mais apropriada para configurar uma determinada rede depende de alguns factores, como por exemplo do tipo de tráfego ou das aplicações suportadas na rede.

9. Conclusões

Este capítulo apresenta as conclusões gerais deste trabalho. Os objectivos definidos para esta dissertação foram na sua generalidade atingidos. Um conjunto de problemas de desempenho em redes de comunicações foi estudado com o auxílio do simulador de redes NS.

Ao longo deste trabalho foi adquirido conhecimento da plataforma seleccionada para efectuar as simulações, o NS. Nesta dissertação foram analisados os conceitos básicos do simulador necessários à criação, simulação e análise de cenários com características específicas. O conhecimento adquirido sobre o NS ao longo desta dissertação foi compilado em anexos, que se constituem em pequenos manuais práticos do NS, úteis na exploração desta dissertação para fins pedagógicos.

O primeiro aspecto de desempenho analisado, que surge naturalmente quando se estudam redes de comunicações, centrou-se nos sistemas de filas de espera. Os sistemas analisados por simulação consistiram no M/M/1, M/D/1, M/G/1 e M/G/1 com prioridades. Estes modelos permitem calcular analiticamente parâmetros de desempenho importantes das redes tais como o atraso médio e o número médio de pacotes na fila de espera e no sistema. Os cenários de simulação escolhidos para cada sistema permitiram a consolidação dos princípios teóricos dos sistemas de filas de espera, e os resultados obtidos permitiram validar os valores teóricos calculados para cada sistema. Foi também efectuada uma análise da validade da aproximação de Kleinrock. Dos resultados obtidos foi possível concluir que esta aproximação é boa quando a ocupação da ligação não é muito elevada. Como parte deste trabalho foram efectuadas extensões ao NS, nomeadamente a criação de dois novos objectos necessários para simulação de um sistema M/M/1.

O segundo aspecto estudado foi o desempenho dos algoritmos de escalonamento. Os algoritmos de escalonamento estudados e comparados por simulação foram FIFO, SFQ, DRR e CBQ. Do estudo efectuado verificou-se que os algoritmos com mais do que uma fila de espera (SFQ, DRR e CBQ) permitem isolar fluxos de tráfego “mal comportado” dos restantes fluxos de tráfego. Os algoritmos que não têm em conta o tamanho dos pacotes (SFQ e CBQ) não permitem uma partilha equitativa da largura de banda entre fluxos de tráfego que transmitem à mesma taxa mas com tamanho de pacotes distintos em cada fluxo. Dos quatro

algoritmos estudados, o DRR é o que melhor se adapta às diversas situações simuladas por ter em conta o tamanho dos pacotes.

Ao nível do descarte de pacotes em situações de congestionamento foram analisadas as técnicas de descarte DropTail, DropFront e RED. As duas primeiras descartam pacotes das suas filas de espera sempre que não têm espaço disponível para receber o pacote que acaba de chegar. No caso da técnica de descarte RED são descartados aleatoriamente os pacotes que chegam, de acordo com o tamanho da fila de espera. Através de simulação, foi possível visualizar o princípio de funcionamento da técnica de descarte RED, que controla o tamanho médio da fila de espera, estabilizando esse valor entre o valor do limiar inferior e do limiar superior configurados. O efeito de sincronização global foi analisado por simulação. A eliminação deste efeito foi testada quando utilizada a técnica de descarte RED, dado que os fluxos são notificados aleatoriamente para diminuírem o tamanho das suas janelas, através do descarte de um dos seus pacotes.

A evolução do protocolo de transporte TCP foi abordada nesta dissertação, começando por ordem cronológica pela primeira versão especificada no RFC793, passando pela Tahoe, de seguida pela Reno e finalmente a Vegas. No estudo experimental efectuado foi possível visualizar, através de gráficos da evolução temporal da janela de congestionamento e da largura de banda utilizada, o efeito dos mecanismos de controlo de congestionamento introduzidos em cada versão do TCP. Num cenário de uma rede congestionada onde é pretendido enviar uma determinada informação durante um determinado período, a necessidade dos mecanismos de controlo de congestionamento é bem visível quando a utilização do TCP RFC793 não permite a transmissão da informação dentro do intervalo de tempo pretendido. A implementação no TCP Tahoe dos primeiros mecanismos de congestionamento, *Slow Start* e *Congestion Avoidance*, permite no mesmo cenário a transmissão da informação dentro do intervalo de tempo pretendido. O novo mecanismo de retransmissão do TCP Tahoe, designado *Fast Retransmit*, permite um melhor aproveitamento da largura de banda ao eliminar os tempos mortos provocados pelo método de detectar as perdas de pacote por *timeout*. O TCP Reno implementa o mecanismo *Fast Recovery* como um complemento do mecanismo *Fast Retransmit*, que evita quebras acentuadas na ocupação da largura de banda. O TCP Vegas altera os mecanismos *Congestion Avoidance*, *Slow Start* e *Fast Retransmit* presentes nas versões anteriores, permitindo com essa transformação utilizar mais eficientemente a largura de banda disponível na ligação. Na comparação, efectuada por simulação, das duas últimas versões

conclui-se que: em cenários bem dimensionados, a escolha sobre a versão de TCP a utilizar deverá recair sobre o TCP Vegas, que permite um melhor aproveitamento da largura de banda; nos cenários em que a largura de banda tem que ser conquistada, o TCP Reno será a melhor opção. O seu método mais agressivo para estimar a largura de banda disponível ajudará a garantir uma fracção da largura de banda disponível.

Os mecanismos de controlo da taxa de transmissão do tráfego que entra na rede são necessários de modo a garantir as taxas contratadas. Estes mecanismos podem ser de policiamento ou de formatação de tráfego dependendo das necessidades de configuração da rede. Nesta dissertação foram estudados os mecanismos de policiamento, começando pelo mais conhecido, o *Token bucket*, passando pelo srTCM e trTCM (ambos baseados no *Token bucket*) e finalmente o tswTCM. Os mecanismos de policiamento utilizam cores para marcar os pacotes, usualmente o verde, o amarelo e o vermelho. O estudo experimental efectuado permitiu consolidar o conhecimento sobre o princípio de funcionamento de cada mecanismo. A escolha sobre qual o mecanismo a escolher depende principalmente do que se pretende: *Token bucket* para controlar uma determinada taxa e a rajada máxima de pacotes aceite; srTCM para controlar uma determinada taxa e dois limiares do tamanho da rajada de pacotes aceite; trTCM para controlar duas determinadas taxas e a rajada máxima de pacotes de cada taxa aceite; tswTCM para controlar duas determinadas taxas.

A evolução das redes de comunicações possibilitou o surgimento de novos serviços e aplicações que necessitam de garantias de QoS por parte da rede, de modo a funcionarem com a qualidade necessária, como por exemplo valores reduzidos do rácio de pacotes perdidos, ou do atraso médio dos pacotes na rede, ou larguras de banda mínimas. O IETF formou grupos de trabalho que definiram a implementação da QoS. Destes grupos de trabalho surgiram duas arquitecturas distintas, a IntServ e a DiffServ. A primeira arquitectura a surgir foi a IntServ, baseada num protocolo de reserva de recursos designado por RSVP. A segunda surgiu devido à limitação da IntServ ser orientada ao fluxo. Na arquitectura DiffServ existe um número limitado de classes de serviço, indicadas no cabeçalho IP de cada pacote (no campo DSCP). Neste trabalho foi explorado por simulação a implementação da QoS num cenário simples utilizando a arquitectura DiffServ com PHB AF e com PHB EF. Na comparação do PHB AF com o serviço *Best-effort* (serviço presente na Internet tradicional) constatou-se o efeito da implementação da QoS, onde um determinado fluxo de tráfego considerado fulcral recebe da rede um tratamento especial. A configuração da arquitectura DiffServ com um PHB EF

permitiu visualizar, em comparação com o PHB AF, que nenhum pacote classificado como pertencente à classe do fluxo de tráfego fulcral foi descartado. A decisão sobre qual a arquitectura mais apropriada para configurar uma determinada rede depende de alguns factores, como por exemplo do tipo de tráfego ou das aplicações suportadas na rede.

9.1 Directivas para trabalho futuro

Nesta Dissertação foram estudados diversos aspectos de desempenho das redes de comunicações. Existe ainda muito trabalho de interesse que poderá ser desenvolvido no futuro nesta vasta área da Engenharia de Tráfego, com o apoio do simulador de redes NS.

Uma das possibilidades de trabalho futuro é a análise do encaminhamento. O simulador de redes NS apresenta várias limitações no que diz respeito a este mecanismo. Uma primeira limitação é a ausência de módulos para simular redes com comutação de circuitos. Seria portanto necessário desenvolver módulos específicos para este efeito. Outra limitação reside no tipo de métodos de encaminhamento suportados pelo NS. O simulador suporta apenas encaminhamento estático (rotas que se mantêm fixas ao longo do tempo) e encaminhamento dinâmico (rotas que podem ser modificadas apenas quando se verifica uma alteração na topologia da rede). O simulador não suporta encaminhamento adaptativo. Neste método as rotas podem ser alteradas em função do estado da rede, medido por exemplo através do atraso médio dos pacotes na rede. Neste caso, uma rota entre determinado par origem-destino pode ser mudada de uma zona mais congestionada da rede para outra menos congestionada. Também aqui seria necessário desenvolver módulos adicionais.

Outra possibilidade de trabalho futuro é a exploração do NS para o estudo de cenários de rede mais reais, que envolvam a integração/interacção dos mecanismos básicos estudados ao longo da dissertação. Propositadamente, esta dissertação focou-se nos diversos mecanismos básicos de rede, por forma a possibilitar a análise e compreensão do princípio de funcionamento e do impacto no desempenho da rede de cada um deles. Pode agora avançar-se para cenários mais complexos, na sequência do que foi já iniciado no capítulo 8 sobre Qualidade de Serviço, onde foi considerada a interacção entre os mecanismos de escalonamento, policiamento e descarte de pacotes.

10. Referências

- [1] The network Simulator, NS-2, <http://nsnam.isi.edu/nsnam/index.php>
- [2] Rui Valadas, Apontamentos das aulas teóricas de Engenharia de Tráfego, do mestrado em Engenharia Electrónica e Telecomunicações, DETUA, 2003/2004
- [3] S. Keshav, “An Engineering Approach to Computer Networking”. Addison-Wesley, 1997
- [4] Nagle, “On Packet Switches with Infinite Storage”, RFC 970, 1987
- [5] A. Demers, S. Keshav and S. Shenker, “Analysis and Simulation of a Fair Queueing Algorithm”, Proceedings of the ACM SIGMOMM’89
- [6] P. McKenney, “Stochastic Fairness Queueing”, Proceedings of INFOCOM’90
- [7] M. Shreedhar and G. Varghese, “Efficient Fair Queueing using Deficit Round Robin”, in Proc. SIGCOMM, Boston, MA, Aug. 1995
- [8] Sally Floyd and Van Jacobson, “Random Early Detection Gateways for Congestion Avoidance”, IEEE/ACM. Transactions on Networking, V.1 N.4, Aug. 1993
- [9] RFC793, “Transmission Protocol Darpa Internet Program Protocol Specification”, Sep 1981
- [10] RFC2581, “TCP Congestion Control” (congestion-Start, Congestion Avoidance, Fast Retransmit, Fast Recovery), Apr. 1999.
- [11] V. Jacobson, "Congestion Avoidance and Control", Computer Communication Review, vol. 18, no. 4, pp. 314-329, Aug. 1988
- [12] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm", end2end-interest mailing list, April 30, 1990 (Fast Recovery)
- [13] L.S. Brakmo and L.L. Peterson, “TCP Vegas: end to end congestion avoidance on a global Internet”, IEEE Journal on Selected Areas in Communications, 13 (8) :1465-80, October 1995
- [14] Andrew S. Tanenbaum, “Computer Networks”. Prentice Hall, 1996
- [15] Larry L. Peterson & Bruce S. Davie, “Computer Networks a System Approach, 1st ed.”. Addison-Wesley, 1996

- [16] Lewis Mackenzie, “Communications and Networks”. Mc Graw Hill, 1998
- [17] L.S. Brakmo, S. W. O’Malley and L.L. Peterson, “TCP Vegas: New Techniques for Congestion Detection and Avoidance”, 1994 ACM SIGCOMM Conference, pages 24-35, May 1994
- [18] Sarut Vanichpun and Wu-chun Feng, “On the Transient Behavior of TCP Vegas”, in Proceedings of ICCCN 2002, Miami, Oct 2002
- [19] Steven H. Low, L. L. Peterson and Limin Wang, “Understanding Vegas: aduality model”, J. ACM, vol. 49, no. 2, pp 207-235, Mar 2002
- [20] U. Hengartner, J. Bolliger and Th. Gross, “TCP Vegas Revisited”, in Proceedings of IEEE Infocom 2000, pp. 1546-1555, Mar 2000
- [21] <http://www.Internet.gov.pt/inbl/iniciativa.asp>
- [22] RFC1633, “Integreted Services in the Internet Architecture”, Jun. 1994
- [23] RFC2211, “Specification of the Controlled-Load Network Element Service”, Sep. 1997
- [24] RFC2212, “Specification of Guaranteed Quality of Service”, Sep. 1997
- [25] RFC2597, “Assured Forwarding PHB Group”, Jun. 1999
- [26] RFC3260, “New Terminology and Clarifications for DiffServ”, Apr. 2002.
- [27] RFC2598, “An Expedited Forwarding PHB”, Jun. 1999.
- [28] RFC3246, “An Expedited Forwarding PHB (Per-Hop Behavior), Mar. 2002.
- [29] RFC2697, “A Single Rate Three Color Marker”, Set. 1999.
- [30] RFC2698, “A Two Rate Three Color Marker”, Set. 1999.
- [31] RFC2859, “A Time Sliding Window Three Colour Marker (TSWTCM)”, Jun. 2000.

- [32] R. Makkar, I. Lambadaris, J. Salim, N. Seddigh, B. Nandy, J. Babiarez, “Empirical Study of Buffer Management Scheme for DiffServ Assured Forwarding PHB, ICCCN2000.
- [33] RFC2475, “ An Architecture for Differentiated Services”, Dec 1998.
- [34] D.D. Clark, W. Fang, “Explicit Allocation of Best Effort Packet Delivery Service”, IEEE/ACM Transactions on Networking, August 1998, Vol 6. No. 4, pp.362-373.
- [35] Sally Floyd and Van Jacobson, “Link-sharing and Resource Management Models for Packet Networks”, IEEE/ACM. Transactions on Networking, V.3 N.4, Aug. 1995

11. Apêndices

A. Instalação e Documentação do NS

O NS foi desenvolvido inicialmente em UNIX, mas também é possível instalá-lo numa plataforma *Windows* através do *Cygnin*.

A última versão do *software* é ns-30 (*released Sept 26, 2006*) e está disponível através da página principal: <http://www.isi.edu/nsnam/ns/ns-build.html>.

O NS pode ser instalado de duas formas: o *allinone* ou o *in pieces*. O *allinone* é um pacote único que requer aproximadamente 250 MB de espaço de disco, e é o aconselhável para quem se inicia no NS. O *in pieces* como o próprio nome indica permite instalar apenas os componentes que se deseja, libertando espaço em disco e é o utilizado por quem efectua desenvolvimento.

O pacote de *software*, designado por *ns-allinone*, que contém os seguintes programa:

- *Tcl release 8.4.11* (necessário)
- *Tk release 8.4.11* (necessário)
- *OTcl release 1.12* (necessário)
- *TclCL release 1.18* (necessário)
- *Ns release 2.30* (necessário)
- *Nam release 1.12* (opcional)
- *Xgraph version 12.1* (opcional)
- *CWeb version 3.4g* (opcional)
- *SGB version 1.0 (?)* (opcional)
- *Gt-itm gt-itm and sgb2ns 1.1* (opcional)
- *Zlib version 1.1.4* (opcional, necessário se o NAM for utilizado)

Para instalar no UNIX:

- cd into the OTcl directory
- run *./configure*
- run *make*
- cd into the TclCL directory
- run *./configure*
- run *make*
- cd into the ns directory
- run *./configure*
- run *make*

Para instalar no windows:

- Instalar o *cygwin*, *software* disponível na página: <http://www.cygwin.com/>
- Seguir um tutorial criado por Nicolas Christin:
<http://www.sims.berkeley.edu/~christin/ns-cygwin.shtml>

Depois de instalado é necessário correr os testes de validação, de modo a verificar se o programa foi correctamente instalado.

- cd ns-2.28; *./validate*

O NS disponibiliza na Internet uma página dedicada a identificação e resolução de problemas ocorridos durante a instalação:

<http://www.isi.edu/nsnam/ns/ns-problems.html>

Duas novas versões do NS2 surgiram no decorrer deste trabalho, optou-se por não efectuar o *upgrade* porque as alterações efectuadas não são relevantes para a execução do mesmo.

Este trabalho é realizado com a versão ns-2.26.

Documentação

O NS está em constante desenvolvimento, com cerca de duas novas versões por ano, o que torna difícil ter documentação actualizada.

A página principal da aplicação é <http://www.isi.edu/nsnam/ns/>.

Através da página principal é possível consultar entre muito outras coisas:

- O manual de NS (<http://www.isi.edu/nsnam/ns/ns-documentation.html>);
- Um tutorial (<http://www.isi.edu/nsnam/ns/tutorial/index.html>);
- Pesquisar a *mail list* (<http://www.isi.edu/nsnam/htdig/search.html>);
- *FAQ* (<http://www.isi.edu/nsnam/ns/ns-faq.html>);

Pesquisando na Internet encontra-se uma grande quantidade de documentação sobre o NS, de salientar um tutorial muito interessante designado por *NS by Example* (<http://nile.wpi.edu/NS/>).

B. Configurações do NS

Uma simulação no NS começa sempre com o seguinte comando:

```
set ns [new Simulator]
```

Usualmente é a primeira linha de código do *script* Tcl. Este comando cria um objecto *Simulator* através do qual todos os outros objectos vão ser instanciados. O objecto *Simulator* vai controlar e efectuar a simulação.

B.1. Nó

Cada nó contém a seguinte informação:

- o Um endereço ou *id_* começa em zero e é incrementado por cada nó criado;
- o Uma listagem dos nós vizinhos (*neighbor_*);
- o Uma listagem dos agentes associados ao nó (*agent_*);
- o Tipo de nó (*Unicast* ou *Multicast*);
- o Módulo de encaminhamento, onde são colocados os classificadores específicos de cada protocolo de encaminhamento.

O comando utilizado para criar os nós é:

```
set node0 [$ns node]  
set node1 [$ns node]
```

Localização do objecto C++: ...\\ns-allinone-2.26\\ns-2.26\\common**node.cc**

B.2. Ligação

Para criar uma ligação bidireccional entre os dois nós criados utiliza-se o seguinte comando:

```
$ns duplex-link $node0 $node1 <bandwidth> <delay> <queue_type>
```

O comando cria uma ligação entre o nó *node0* e o *node1* com uma largura de banda igual ao valor de <bandwidth>, um atraso de propagação igual ao valor de <delay> e com uma fila de espera do tipo <queue_type>. Para criar uma ligação unidireccional entre o *node0* e o *node1* apenas é necessário substituir *duplex-link* por *simplex-link*.

As filas de espera pode ser configurada do tipo DropTail, FQ, SFQ, DRR ou RED.

1. DropTail

Variáveis acessíveis no *script* TCL (a negrito estão os valores por defeito):

- drop_front_ **[FALSE]** permite alterar a técnica de descarte de pacote de DropTail para DropFront;
- queue_in_bytes_ **[FALSE]** a fila de espera pode ser medida em pacotes ou em *bytes*;
- mean_pktsize_ **[500]** tamanho médio dos pacotes em *bytes*, valor necessário se optar por medir a fila de espera em *bytes*;
- summarystats_ **[FALSE]** activada permite registrar o tamanho médio da fila de espera.

2. FQ

Variáveis acessíveis no *script* TCL (a negrito estão os valores por defeito):

- SecsPerByte_ **[0]** variável utilizada para definir o tempo em segundos para enviar um *byte*.

O único parâmetro acessível via *script* TCL é utilizado num ficheiro Tcl de inicialização (ns-2.26/tcl/lib/ns-queue.tcl), onde é afectado pela seguinte expressão:

$$\text{secsPerByte_} = 8 / LB$$

Onde LB é a largura de banda da ligação em bits/s.

O valor anterior multiplicado pelo tamanho do pacote em *bytes*, $\text{size}(P_k^i)$, dá em segundos o tempo necessário para enviar o pacote, valor indicado na variável delta_ .

$$\text{delta_} = \text{size} \times \text{secsPerByte_}$$

Na implementação do NS os valores S_k^i e F_k^i estão representados pela mesma variável finishTime_ .

O valor finishTime_ dentro da função $\text{recv}()$ representa S_k^i e é calculado segundo a seguinte expressão enunciada no conceito teórico do algoritmo:

$$S_k^i = \text{MAX}(F_{k-1}^i, R(t_k^i))$$

Ou seja, quando chega o primeiro pacote k a fila i no instante t_k^i , o valor de S_k^i é igual a t_k^i . Para os seguintes pacotes se $t_k^i > F_{k-1}^i$, então S_k^i é igual a t_k^i , senão é igual a F_{k-1}^i .

Na função `deque()` de modo a decidir qual o pacote a enviar, é calculado F_k^i através da função `finish()` utilizando a seguinte expressão:

$$F_k^i = S_k^i + P_k^i \times \text{secsPerByte_} \times \text{nactive}$$

Depois de enviado o pacote escolhido a variável `finishTime_` da fila é afectada com o valor F_k^i .

A unidade do tamanho do pacote na implementação do NS é o *byte*, ou seja é feita uma adaptação a disciplina BR emulada para uma *byte-a-byte round Robin*.

Comparando esta expressão da função `deque()` para o cálculo de F_k^i , com a enunciada no conceito teórico verifica-se que foi adicionado a parcela a negrito. O *finish Time* F_k^i tem em conta a largura de banda da ligação e o número de filas activas, calculando um valor mais real para o instante do fim da transmissão do pacote numa disciplina emulada *byte-a-byte Round Robin*.

O valor de `maxflow_` é igual ao maior número de `fid` atribuído para os fluxos, por exemplo se configuramos o fluxo 1 com `fid=0` e o fluxo 2 com `fid=10`, a variável `maxflow_` é igual a 10. Existindo um desperdício de processamento porque o algoritmo vai pesquisar uma a uma as 11 filas, em vez de verificar apenas as 2 existentes.

No NS não é possível configurar o tamanho das filas de espera deste algoritmo. Nas simulações não existe qualquer utilidade prática comparar este algoritmo com os restantes, uma vez que a fila de espera do FQ possui tamanho infinito.

Para além dessa limitação que por si só justificava não comparar este algoritmo com os restantes, existe outra limitação. O algoritmo só funciona no NS quando a função de registo da actividade da rede está activa:

```
set tr [open out_sched_fq.tr w]
$ns trace-all $tr
```

Esse registo para além de pesar no processamento da simulação, cria para o cenário escolhido um ficheiro de tamanho desmedido.

Nome	Tamanho	Tipo
file_f1.stat	35 KB	Ficheiro STAT
file_f2.stat	35 KB	Ficheiro STAT
file_fTotal.stat	36 KB	Ficheiro STAT
mk1-v4.tcl	5 KB	Ficheiro TCL
ns.exe.stackdump	2 KB	Ficheiro STACKDUMP
schedule-v1.tcl	7 KB	Ficheiro TCL
schedule-v2.tcl	8 KB	Ficheiro TCL
xgraph	7 KB	Ficheiro PS
schedule-v3.tcl	8 KB	Ficheiro TCL
out_sched_fq	114.137 KB	Ficheiro TR

3. SFQ

Variáveis acessíveis no *script* TCL (a negrito estão os valores por defeito):

- `maxqueue_ [40]` número máximo de pacotes nas filas de espera;
- `buckets_ [16]` nº máximo de filas de espera.

4. DRR

Variáveis acessíveis no *script* TCL (a negrito estão os valores por defeito):

- `buckets_ [10]` nº máximo de filas de espera;
- `blimit_ [25 000]` tamanho em *bytes* do buffer partilhado pelos fluxos;
- `quantum_ [250]` limiar em *bytes* para cada fila;
- `mask_ [0]` Quando colocado a 0 significa que um fluxo é consiste nos pacotes enviados pelo mesmo nó, podendo ser por portos diferentes, se for colocado a 1 um fluxo é identificado pela identificação do nó e do respectivo porto.

5. CBQ

Os parâmetros configuráveis no NS são:

- `$cbq insert <class>` - insere a classe de tráfego a estrutura partilhada do link associada a ligação CBQ;
- `$cbq bind <cbqclass> <id1> [$id2]` - Os pacotes que contém a identificação *id1* do fluxo (ou no intervalo de *id1* até *id2* inclusivé) são associados a classe CBQ*classe*;
- `$cbq algorithm <alg>` - os possíveis algoritmos são: *ancestor-only*, *top-level*, ou *formal*, por defeito o valor é 0;
- `maxpkt_` -o tamanho máximo do pacote em *bytes*. É apenas utilizado pelo objecto CBQ/*WRR* para calcular a máxima largura de banda para o *WFQ*, por defeito é 1024;

Para cada classe é necessário:

Criar a classe:

- `set class1 [new CBQClass]`

Criar a fila de espera:

- `set queue1 [new Queue/DropTail]`

Associar a fila a classe:

- `$class1 install-queue $queue1`

Definir os parâmetros da classe:

- `$class1 setparams parent okborrow_ allot_ maxidle_ priority_ level_ xdelay`

Os parâmetros são:

- `parent_` - indica a classe a cima;
- `okborrow_` - este boolean indica que uma classe pode pedir emprestado largura de banda de uma classe parente, por defeito é *true*;
- `allot_` - é a máxima porção de largura de banda atribuída a classe e é um número real entre 0,0 e 1,0;
- `maxidle_` - é o tempo máximo que uma classe pode pedir que os seus pacotes fiquem em fila de espera antes de serem transmitidos;
- `priority_` - é a prioridade da classe. O valor pode ser de 0 a 10, e mais que uma classe pode ter a mesma prioridade. Prioridade 0 é a mais prioritária;
- `level_` - é o nível da classe na partilha da ligação hierárquizada.
- `xdelay_` - é o atraso possível de introduzir na fila.

6. RED

Variáveis acessíveis no *script* TCL (a negrito estão os valores por defeito):

- `bytes_ [FALSE]` se *TRUE* activa o *byte-mode* do RED, onde o tamanho do pacote que chega influencia a probabilidade do pacote ser descartado (ou marcado). No código C++ é representado pela variável *edp_.bytes*;
- `queue_in_bytes_ [FALSE]` se *TRUE* o valor média da fila de espera é calculada em *bytes* e não em pacotes.
- `thresh_ [0]` limiar inferior para o número médio de pacotes na fila de espera
- `maxthresh_ [0]` limiar superior para o número médio de pacotes na fila de espera
- `mean_pktsize_ [500]` é o valor estimado para o tamanho médio do pacote. Usado para actualizar o valor médio calculado da fila após um período em vazio.
- `idle_pktsize_ [100]` é o valor estimado para o tamanho médio do pacote. Usado para actualizar o valor médio calculado da fila num período em vazio.
- `q_weight_ [-1]` utilizado para calcular o valor médio do tamanho da fila de espera
- `wait_ [TRUE]` para manter o intervalo entre os pacotes descartados
- `linterm_ [10]` assim como o tamanho médio da fila varia entre *thresh_* e *maxthresh_* a probabilidade de descartar um pacote varia entre 0 e $1/linterm_$
- `setbit_ [FALSE]` se *TRUE* marca o pacote activando o bit ECN que indica o congestionamento em vez de descartar os pacotes.
- `summarystats_ [FALSE]` activada permite registrar o tamanho médio da fila de espera
- `gentle_ [TRUE]` quando activada aumenta lentamente a probabilidade de descarte quando o tamanho médio da fila excede o limiar superior
- `drop_tail_ [TRUE]` para descartar o último pacote
- `drop_front_ [FALSE]` quando activada em vez de descartar o último descarta o primeiro da fila
- `drop_rand_ [FALSE]` quando activada em vez de descartar o último descarta um aleatoriamente.

Localização dos objectos C++: ...\\ns-allinone-2.26\\ns-2.26\\link\\...
...\\ns-allinone-2.26\\ns-2.26\\queue\\...

B.3. Agente

É possível configurar em todos os agentes os seguintes parâmetros:

- `fid_` Identifica um fluxo;
- `prio_` Define uma prioridade;
- `agent_addr_` Indica qual o endereço deste agente;
- `agent_port_` Indica qual o porto de saída deste agente;
- `dst_addr_` Indica qual o endereço do agente de destino;
- `dst_port_` Indica qual o porto de saída do agente de destino;
- `ttl_` Valor de TTL (32 por defeito).

Comando utilizado para configurar um dos parâmetros:

```
$src0 set fid_ 0
```

Localização do objecto C++: ...\\ns-allinone-2.26\\ns-2.26\\common**agent.cc**

Dependendo do tipo de protocolo de transporte a utilizar existem vários agentes de origem e de destino, com parâmetros específicos a cada um. O comando para associar o agente ao nó e o de conectar os agentes são sempre os mesmos, independentemente dos agentes escolhidos.

UDP – Agente de origem

Apenas existe um tipo de agente UDP origem, que é construído através do seguinte comando:

```
set src0 [new Agent/UDP]
```

Localização do objecto C++: ...\\ns-allinone-2.26\\ns-2.26\\apps**udp.cc**

Para além dos parâmetros descritos para os agentes, no caso do UDP existe outro parâmetro que define o tamanho do pacote em *bytes*.

- `packetSize_` Tamanho dos pacotes em *bytes* (por defeito é igual a 1000 *bytes*);

Utiliza-se a seguinte linha de código para alterar o tamanho dos pacotes:

```
$src0 set packetSize_ 3000
```

UDP – Agente de destino

Existem no NS dois tipos de agentes destino para UDP e foi criado ao longo desta dissertação um novo tipo (ver capítulo 3).

1. *Null Agent*

```
set sink0 [new Agent/Null]
```

Este agente apenas recebe os pacotes e descarta-os.

2. *Loss monitor*

```
set sink0 [new Agent/LossMonitor]
```

Localização do objecto C++: ...\\ns-allinone-2.26\\ns-2.26\\tools**loss-monitor.cc**

Este agente possui contadores estatísticos que permitem calcular a quantidade de pacotes e de *bytes* que chegam do nó origem. Através do número de sequência de cada pacote calcula o número de pacotes descartados. É também possível saber o instante em segundos que chegou o último pacote e qual o próximo pacote esperado, indicado pelo seu número de sequência.

Variáveis de estado:

- `nlost_` Número de pacotes descartados;
- `npkts_` Número de pacotes recebidos;
- `bytes_` Número de *bytes* recebidos;
- `lastPktTime_` Instante de tempo do último pacote recebido;
- `expected_` Número de sequência do próximo pacote

3. *Node Monitor*

```
set sink0 [new Agent/NodeMonitor]
```

Localização do objecto C++: ...\\ns-allinone-2.26\\ns-2.26\\tools**node-monitor.cc**

Este novo agente implementa todas as funcionalidades do agente anterior, altera o método para o cálculo de pacotes perdidos e possui novos contadores estatísticos. As especificações deste novo objecto estão descritas no capítulo 3.

Variáveis de estado:

- `maxDelay_` Valor máximo dos atrasos dos pacotes;
- `minDelay_` Valor mínimo dos atrasos dos pacotes;

Para além dos dois novos parâmetros, foi adicionada ao objecto uma nova funcionalidade, a capacidade de calcular o atraso médio dos pacotes.

- Primeiro é necessário criar um objecto do tipo *Samples*:
`set delaytotal [new Samples]`
- Associar esse objecto ao agente *NodeMonitor*:
`$sink0 set-delay-samples $delaytotal`
- Activar a recolha dos atrasos para o objecto criado:
`set delayst [$sink0 get-delay-samples]`
- A variável *delayMeanTotal* é igual ao atraso médio dos pacotes
`set delayMeanTotal [$delayst mean]`

TCP – Agente de origem

Existem implementados no NS várias versões de agente TCP origem:

Localização do objecto C++ do TCP: ...\\ns-allinone-2.26\\ns-2.26\\tcp\\...

1. Tahoe

```
set src1 [new Agent/TCP]
```

Parâmetros configuráveis:

- `window_` Limite superior da *advertised window* (20 por defeito);
- `ecn_` Quando igual a 1 é utilizado o bit ECN para assinalar o congestionamento na rede (0 por defeito);
- `packetSize_` Tamanho em *bytes* de todos os pacotes deste agente (1000 *bytes* por defeito);

- `tcip_base_hdr_size_` Tamanho em *bytes* do cabeçalho dos pacotes TCP (40 *bytes* por defeito);
- `ts_option_size_` Activando a opção timestamp adiciona ao cabeçalho mais uns *bytes* (10 *bytes* por defeito);
- `timestamps` Quando activa adiciona ao valor de `tcip_base_hdr_size_` o valor de `ts_option_size_` (false por defeito)

Variáveis de estado:

- `ack_` Número de ACK recebidos;
- `dupacks_` Número de ACK duplicados;
- `maxseq_` *sequence number* mais elevado dos dados transmitidos;
- `ndatapack_` Número de pacotes transmitido (se um pacote for transmitido várias vezes, aqui só é contado uma vez);
- `ndatabytes_` Número de *bytes* transmitido;
- `nrexmit_` Número de pacotes devido a timeouts;
- `nrexmitpack_` Número de pacotes retransmitidos;
- `nrexmitbytes_` Quantidade de *bytes* retransmitidos;
- `cwnd_` Valor da *congestion window*;
- `ncwndcuts_` Número de vezes que `cwnd_` foi reduzida;
- `necnresponses_` Número de vezes que `cwnd_` foi reduzida por receber um pacote *ECN*.
- `ssthesh_` Valor do limite superior do *Slow Start*;
- `rtt_` Valor estimado do *Round-trip time*;
- `rttvar_` Valor estimado médio do desvio do *Round-trip time*;

Os parâmetros e as variáveis de estado descritos podem ser utilizados em qualquer versão de TCP. Algumas dessas versões têm parâmetros e/ou variáveis de estado específicos.

2. Reno

```
set src1 [new Agent/TCP/Reno]
```

Não tem parâmetros ou variáveis de estado próprios.

3. **NewReno**

```
set src1 [new Agent/TCP/Newreno]
```

Não tem parâmetros ou variáveis de estado próprios.

4. **Vegas**

```
set src1 [new Agent/TCP/Vegas]
```

Parâmetros configuráveis:

- `v_alpha_` Limiar inferior em pacotes (1 por defeito);
- `v_beta_` Limiar superior em pacotes (3 por defeito);
- `v_gamma_` Limiar em pacotes para passar do modo *Slow Start* para o modo *congestion avoidance* (1 por defeito);
- `v_rtt_` (0 por defeito).

5. **Sack1**

```
set src1 [new Agent/TCP/Sack1]
```

Não tem parâmetros ou variáveis de estado configuráveis.

6. **FACK**

```
set src1 [new Agent/TCP/FACK]
```

Parâmetros configuráveis:

- `ss-div4` Divide o *ssthresh* por quatro em vez de por dois, quando o congestionamento é detectado num período $\frac{1}{2}$ RTT do *Slow Start*. (1 = *Enable*, 0 = *Disable*);
- `rampdown` Reduz devagar a *congestion window* em vez de a reduzir instantaneamente em metade o seu valor. (1 = *Enable*, 0 = *Disable*)

TCP – Agente de destino

No TCP o agente de destino tem um papel activo no protocolo, gerando para cada pacote recebido uma confirmação (ACK), enviando esse ACK ao nó de origem de modo a garantir que toda informação transmitida pela origem chegue ao nó destino.

1. TCPSINK

```
set sink1 [new Agent/TCPSink]
```

Parâmetros configuráveis:

- `packetSize_` Tamanho em *bytes* do ACK (40 *bytes* por defeito);
- `maxSackBlocks_` Número máximo de blocos de informação que podem ser *acknowledged* na opção *SACK*. (3 por defeito).

2. TCPSINK/DELACK

```
set sink1 [new Agent/TCPSink/DelAck]
```

Parâmetros configuráveis:

- `interval_` porção de tempo (atraso) que é necessário aguardar antes de enviar o pacote ACK do pacote recebido. Se entretanto receber outro pacote envia imediatamente o ACK (100ms por defeito).

3. TCPSINK/SACK1

```
set sink1 [new Agent/TCPSink/Sack1]
```

Não possui parâmetros ou variáveis de estado configuráveis.

4. TCPSINK/SACK1/DELACK

```
set sink1 [new Agent/TCPSink/Sack1/DelAck]
```

Parâmetros configuráveis:

- `interval_` porção de tempo (atraso) que é necessário aguardar antes de enviar o pacote ACK do pacote recebido. Se entretanto receber outro pacote envia imediatamente o ACK (100ms por defeito).

Depois de criado é necessário associar o agente de origem ao respectivo nó:

```
set src0 [new Agent/UDP]
$ns attach-agent $node0 $src0
```

Efectuar o mesmo procedimento com o agente de Destino:

```
set sink0 [new Agent/Null]
$ns attach-agent $node1 $sink0
```

Após ter criado os agentes e de os ter associado aos respectivos nós, só falta liga-los através do seguinte comando:

```
$ns connect $src0 $sink0
```

B.4. Geradores de Tráfego e Aplicações

A classe mãe responsável por gerar o tráfego é a classe *TrafficGenerator* (`.\ns-2.26\tools\trafgen.cc`).

1. *EXPOO_Traffic*

Localização do objecto C++: `.\ns-2.26\tools\expoo.cc`

Corresponde à classe OTcl: **Application/Traffic/Exponential**.

As variáveis caracterizáveis são:

- `packetSize_` Tamanho fixo em *bytes* dos pacotes gerados (210 por defeito);
- `burst_time_` Tempo médio a *ON* da fonte (0,5 por defeito);
- `idle_time_` Tempo médio a *OFF* da fonte (0,5 por defeito);
- `rate_` Taxa de transmissão durante o tempo a *ON* (64kb por defeito).

Uma nova fonte de tráfego deste tipo pode ser criada e parametrizada do seguinte modo:

```
set traffic0 [new Application/Traffic/Exponential]
$traffic0 set packetSize_ 1000
$traffic0 set burst_time_ 500ms
$traffic0 set idle_time_ 500ms
$traffic0 set rate_ 100k
```

Segundo o manual do NS [1], parametrizando esta fonte com a variável `burst_time_` igual a zero e a variável `rate_` com um valor muito elevado, essa comporta-se como um processo de *Poisson*, em que os tempos de chegada seguem uma distribuição exponencial com média definida na variável `idle_time_`.

Analisando a função `init()` do objecto C++, verifica-se que a variável `rate_` com um valor muito elevado garante que a variável `rem_` (que define o número de pacotes no próximo tempo *ON*) tende para zero. Na função `next_interval(int&)` verifica-se que a função garante que pelo menos 1 pacote é enviado no período *ON* com `burst_time_` igual a zero.

```
void EXPOO_Traffic::init()
{
    ...
    interval_ = (double)(size_ << 3)/(double)rate_;
    burstlen_.setavg(ontime_/interval_);
    rem_ = 0;
    if (agent_)
        agent_->set_pkttype(PT_EXP);
}
double EXPOO_Traffic::next_interval(int& size)
{
    double t = interval_;

    if (rem_ == 0) {
        /* compute number of packets in next burst */
        rem_ = int(burstlen_.value() + .5);
        /* make sure we got at least 1 */
        if (rem_ == 0)
            rem_ = 1;
        /* start of an idle period, compute idle time */
        t += Offtime_.value();
    }
    rem_--;

    size = size_;
    return(t);
}
```

2. POO_Traffic

Localização do objecto C++: `.\ns-2.26\tools\pareto.cc`

Corresponde à classe Otcl: **Application/Traffic/Pareto**.

As variáveis caracterizáveis são:

- `packetSize_` Tamanho fixo em *bytes* dos pacotes gerados (210 por defeito);
- `burst_time_` Tempo médio a *ON* da fonte (500ms por defeito);
- `idle_time_` Tempo médio a *OFF* da fonte (500ms por defeito);
- `rate_` Taxa de transmissão durante o tempo a *ON* (64kb por defeito);
- `shape_` A forma usada pela distribuição pareto (1,5 por defeito).

Uma nova fonte de tráfego deste tipo pode ser criada e parametrizada do seguinte modo:

```
set traffic0 [new Application/Traffic/Pareto]
$traffic0 set packetSize_ 1000
$traffic0 set burst_time_ 500ms
$traffic0 set idle_time_ 500ms
$traffic0 set rate_ 100k
$traffic0 set shape_ 1,5
```

3. *CBR_Traffic*

Localização do objecto C++: `.\ns-2.26\tools\cbr_traffic.cc`

Corresponde à classe OTcl: **Application/Traffic/CBR**.

As variáveis parametrizáveis são:

- `packetSize_` Tamanho fixo em *bytes* dos pacotes gerados (210 por defeito);
- `rate_` Taxa de transmissão (448kb por defeito);
- `interval_` (opção) intervalo entre pacotes
- `random_` *Flag* que quando activada introduz no cálculo do próximo intervalo um ruído aleatório. (Por defeito está *OFF*).
- `Maxpkts_` Número máximo de pacotes a enviar (268435456 por defeito).

Uma nova fonte de tráfego deste tipo pode ser criada e configurada do seguinte modo:

```
set traffic0 [new Application/Traffic/CBR]
$traffic0 set packetSize_ 48
$traffic0 set rate_ 64k
$traffic0 set random_ 1
```

4. *TrafficTrace*

Localização do objecto C++: `.\ns-2.26\trace\traffictrace.cc`

Corresponde à classe OTcl: **Application/Traffic/Trace**.

Não tem variáveis parametrizáveis.

5. *EXP_EXP_Traffic*

Nova fonte de tráfego introduzida no decorrer desta dissertação.

Localização do objecto C++: `.\ns-2.26\tools\exp-exp.cc`

Corresponde à classe OTcl: **Application/Traffic/Exp_Exp**.

Variáveis parametrizáveis:

- `packetSize_` Tamanho médio em *bytes* dos pacotes gerados (210 por defeito);
- `intervalTime_` Valor médio do intervalo de tempo entre os pacotes transmitidos seguindo uma distribuição exponencial (1s por defeito);

Variável de estado:

- `numpkts_` Número de pacotes transmitidos.

Uma nova fonte de tráfego deste tipo pode ser criada e parametrizada do seguinte modo:

```
set traffic0 [new Application/Traffic/Exp_Exp]
$traffic0 set packetSize_ 1000
$traffic0 set intervalTime_ 500ms
```

6. *Application/Telnet*

Localização do objecto C++: ...\\ns-2.26\\apps**telnet.cc**

Corresponde à classe OTcl: **Application/Telnet**.

Variável parametrizável:

- `interval_` Valor médio do intervalo de tempo entre os pacotes transmitidos seguindo uma distribuição exponencial (1s por defeito);

Uma nova aplicação deste tipo pode ser criada e caracterizada do seguinte modo:

```
set traffic1 [new Application/Telnet]
```

7. *Application/FTP*

Localização do objecto C++: só implementado no OTcl

Corresponde à classe OTcl: **Application/FTP**

Variável parametrizável:

- `maxpkts_` Número máximo de pacotes gerados pela aplicação.

Funções:

- `start` Inicia o envio de pacotes.
- `stop` Parar o envio de pacotes.
- `produce n` Define o número de pacotes a enviar igual a *n*.
- `producemore n` Incrementa o número de pacotes por *n*.
- `send n` Envia *n bytes*.

Uma nova aplicação deste tipo pode ser criada e parametrizada do seguinte modo:

```
set traffic0 [new Application/FTP]
```

Escolhida a fonte de tráfego ou aplicação a utilizar é necessário associar essa fonte ao respectivo agente, através do seguinte comando:

```
$traffic0 attach-agent $src0
```

A transmissão de dados é iniciada através da função `start()` e interrompida através da função `stop()`.

```
$traffic0 start
$traffic0 stop
```

B.5. Agendar eventos

A simulação é iniciada e findada através dos seguintes comandos:

```
set temposim 10
$ns at $temposim "finish"
$ns run
```

A primeira linha coloca na variável `temposim` o valor 10. A segunda, agenda o procedimento `finish()` para o instante de tempo igual a variável `temposim` e a terceira linha inicia a simulação.

O procedimento `finish()` tem que possuir o evento `exit`, de modo a parar a simulação:

```
proc finish {} {
    exit 0
}
```

Outros eventos, como por exemplo iniciar e parar as fontes de tráfego podem ser agendados com os seguintes comandos:

```
$ns at 0.0 "$scr0 start"
$ns at 1.0 "$scr1 start"
$ns at $temposim "$scr0 stop"
$ns at $temposim "$scr1 stop"
```

Podemos criar um procedimento chamado “`meu_proc`” e agendar esse procedimento para o instante 2 através do seguinte comando:

```
$ns at 2.0 "meu_proc"
```

Dentro do procedimento `meu_proc`, podemos agenda-lo com uma periodicidade igual a 2 instante, ou seja o procedimento é executado de 2 em 2 instantes.

```
Proc meu_proc {}{  
  global ns  
  set now [$ns now]  
  set intervalo 2  
  ...  
  $ns at [expr $now + $intervalo] "meu_proc"  
}
```

C. Correr a simulação

O código do *script* para simular este cenário está no anexo E. O *script* pode ser escrito em qualquer editor de texto e deve ser guardado com extensão .Tcl. (ex:: meu_script.tcl)

Para executar o *script* é necessário ir para o directório onde o ficheiro foi gravado e escrever o comando:

```
ns meu_script.tcl
```

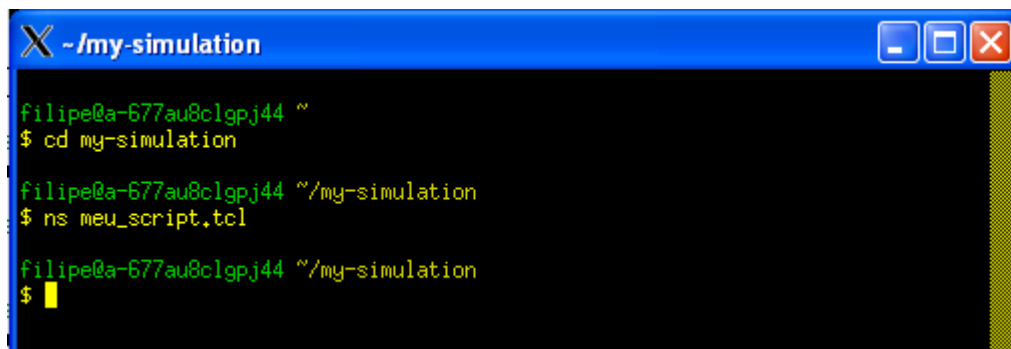


Figura 10.1 – Correr a simulação

Ao correr a simulação, nada acontece!!! O que não é de todo má sinal, porque pelo menos o *script* corre sem erros.

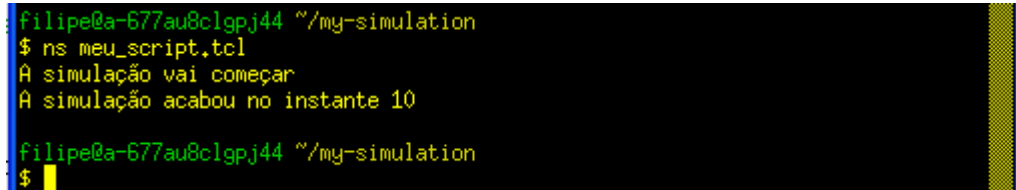
O comando Tcl puts permite visualizar no ecrã uma *string* colocada no seu argumento. Acrescentando as seguintes linhas de comando dentro do procedimento finish() e antes da última linha de código, podemos verificar que a simulação corre.

```

proc finish {} {
    global temposim
    puts "A simulação acabou no instante $temposim"
    exit 0
}

puts "A simulação vai começar"
$ns run
  
```


Obtém-se o seguinte resultado:



```
filipe@a-677au8clgpj44 ~/my-simulation
$ ns meu_script.tcl
A simulação vai começar
A simulação acabou no instante 10
filipe@a-677au8clgpj44 ~/my-simulation
$
```

Figura 10.2 – Correr a simulação com *puts*

É possível confirmar que o tempo real nada tem a ver com o tempo de simulação, o fim da simulação foi agendado para o instante 10 e a simulação não demora mais do que dois segundos a correr.

C.1. Introduzir parâmetros de entrada ao *script*

A primeira linha de código do *script* (`set temposim 10`) foi colocada estrategicamente, de modo a permitir rapidamente localiza-la e alterar o tempo da simulação.

Essa variável, ou outra qualquer pode ser introduzida como parâmetro de entrada do *script* permitindo facilmente correr a simulação para vários valores dessa variável. É necessário efectuar a seguinte alteração ao *script*:

Alterar a linha que afecta a variável `temposim`

```
set temposim 10
```

por:

```
set SINTAX "A syntax é: ns meu_script.tcl <tempo de simulação>"
if {$argc != 1} {puts $SINTAX; exit 0}
```

```
set temposim [lindex $argv 0]
```

A primeira linha afecta a variável `SINTAX` com uma *string* que será utilizada como mensagem de erro. A segunda linha verifica o número de parâmetros de entrada do *script*, se for diferente de 1 então coloca no ecrã (através do comando *puts*) a mensagem de erro. Finalmente a última linha afecta a variável `temposim` com o parâmetro de entrada do *script*.

Utilizando a linha de comando da alínea anterior obtemos o seguinte resultado:

```

filipe@a-677au8clgpj44 ~/my-simulation
$ ns meu_script.tcl
A syntax é: ns meu_script.tcl <tempo de simulação>

filipe@a-677au8clgpj44 ~/my-simulation
$

```

Figura 10.3 – Mensagem de erro

Repetindo mas agora utilizando o parâmetro de entrada:

```

filipe@a-677au8clgpj44 ~/my-simulation
$ ns meu_script.tcl 10
A simulação vai começar
A simulação acabou no instante 10

filipe@a-677au8clgpj44 ~/my-simulation
$

```

Figura 10.4 – Script com parâmetros de entrada

C.2. Repetir a simulação

Para estudar a simulação com vários valores de tempo *sim* (por exemplo para 10, 20 e 30), é necessário correr o *script* três vezes.

```

filipe@a-677au8clgpj44 ~/my-simulation
$ ns meu_script.tcl 10
A simulação vai começar
A simulação acabou no instante 10

filipe@a-677au8clgpj44 ~/my-simulation
$ ns meu_script.tcl 20
A simulação vai começar
A simulação acabou no instante 20

filipe@a-677au8clgpj44 ~/my-simulation
$ ns meu_script.tcl 30
A simulação vai começar
A simulação acabou no instante 30

filipe@a-677au8clgpj44 ~/my-simulation
$

```

Figura 10.5 – Repetir a execução do script para repetir a simulação

OU a alternativa é efectuar a seguinte linha de comando:

```
for var in 10 20 30; do ns meu_script.tcl $var;done
```

```

filipe@a-677au8clgpj44 ~/my-simulation
$ for var in 10 20 30; do ns meu_script.tcl $var;done
A simulação vai começar
A simulação acabou no instante 10
A simulação vai começar
A simulação acabou no instante 20
A simulação vai começar
A simulação acabou no instante 30

filipe@a-677au8clgpj44 ~/my-simulation
$

```

Figura 10.6 – Repetir a simulação

D. Resultados da simulação

D.1. Ler as variáveis de estado dos objectos

Leitura do número de pacotes recebidos para por cada agente de destino no final da simulação.

```
proc finish {} {
  global temposim sink0 src1
  set pacotes_UDP [$sink0 set npkts_]
  set pacotes_TCP [$src1 set ack_]
  puts "Número de pacotes recebidos:"
  puts "UDP: $pacotes_UDP | TCP: $pacotes_TCP"
  puts "A simulação acabou no instante $temposim"
  exit 0
}
```

O agente de destino TCP não tem variáveis de estado mas, como envia uma confirmação (ACK) por cada pacote recebido, é possível contabilizar os pacotes recebidos pelo agente destino contando os ACK recebidos no agente de origem.

Criando um procedimento pode-se analisar o número de pacotes recebidos por exemplo em cada instante, designando-se o novo procedimento por `record()`:

```
proc record {} {
  global ns sink0 src1

  set periodo 1
  set now [$ns now]

  set pacotes_UDP [$sink0 set npkts_]
  set pacotes_TCP [$src1 set ack_]
  puts "Número de pacotes recebidos ao instante $now:"
  puts "UDP: $pacotes_UDP | TCP: $pacotes_TCP"

  $ns at [expr $now+$periodo] "record"
}
```

Agenda-se esse procedimento para o primeiro instante:

```
$ns at 1.0 "record"
```

O evento `record` é executado no instante 1 e depois de executado volta a agendar-se para o instante `now + periodo`, onde a variável `now` é igual ao instante actual da simulação no

momento que o procedimento é executado e `periodo` é a periodicidade escolhida para repetir este evento.

D.2. Identificar as variáveis configuráveis dos objectos

O anexo B não descreve todos os objectos presentes no NS. Uma forma de identificar rapidamente as variáveis, os parâmetros de configuração e as variáveis de estado dos restantes objectos acessíveis no *script* Tcl, é localizar no código do Objecto C++ a função `bind()`. Esta função é responsável pela ligação das variáveis dos objectos C++ ao *script* Tcl, permitindo no *script* ler e/ou alterar o valor dessas variáveis.

No NS existem quatro diferentes funções `bind()`, para cinco diferentes tipos de variáveis:

- `bind()` : Variáveis reais e inteiras;
- `bind_time()` : Variável tempo;
- `bind_bw()` : Variável Largura de Banda;
- `bind_bool()` : Variável Boleana.

A identificação dessas variáveis de um determinado objecto também pode ser consultada no ficheiro responsável pela inicialização de todas essas variáveis:

```
.\ns-2.26\tcl\lib\ns-default.tcl
```

A figura 10.7 apresenta o exemplo de pesquisar as variáveis do objecto *Agent/Lossmonitor*.

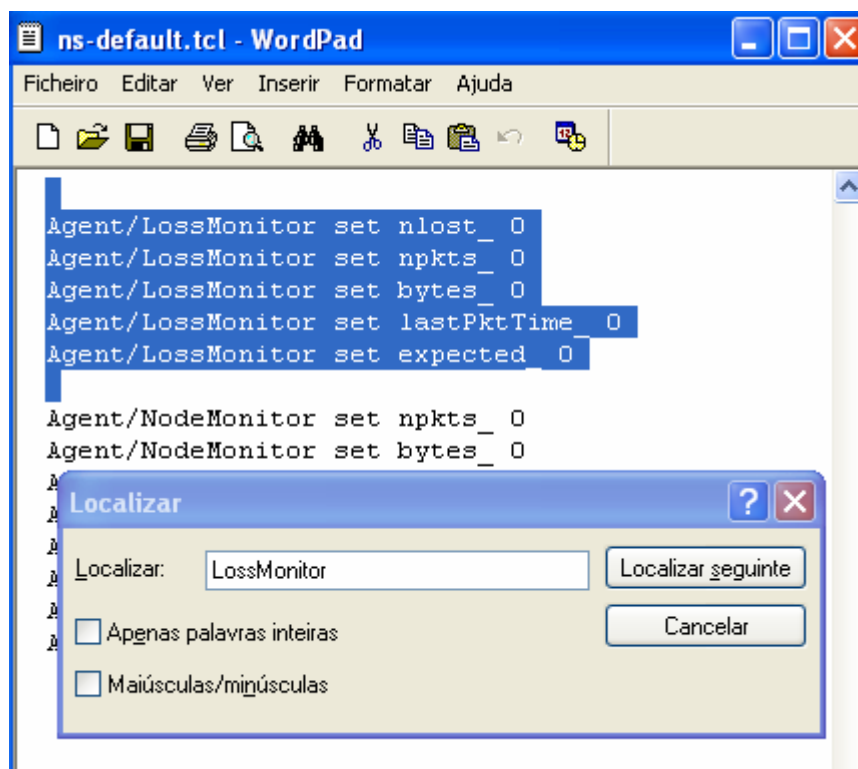


Figura 10.7 - Pesquisa ficheiro ns-default.tcl

D.3. Registo de actividade da rede (*Trace*)

Registos efectuados nas interligações

A passagem dos pacotes pelas ligações é registada e guardada em ficheiro. Para activar essa funcionalidade utiliza-se os seguintes comandos (a negrito) no *script* Tcl:

```
#Criar Objecto Simulator
set ns [new Simulator]
#
# Trace File
#
set tr [open out_script.tr w]
$ns trace-all $tr
...
proc finish {} {
    global temposim sink0 src1 tr ns

    $ns flush-trace
    close $tr
    ...
}
```

A primeira linha a negrito cria um ficheiro no modo de escrita e associa a esse ficheiro o nome lógico tr. O comando trace-all inicia o registo dos eventos dos pacotes no ficheiro criado.

No final da simulação dentro do procedimento `finish()`, o registo da actividade é inactivo através do comando `flush-trace` e o ficheiro deve ser cerrado de modo a ser acessível para leitura.

Em vez de activar o registo de todas as ligações, é possível activar apenas as desejadas substituindo a linha de comando, `ns trace-all $tr`, pela seguinte:

```
$ns trace-queue $node0 $node1 $tr
```

Este comando deve ser escrito após a criação da ligação.

Cada linha do ficheiro criado representa um evento. O formato é o seguinte:

Event	Time	From node	To node	Pkt type	Pkt size	Flags	Fid	Src addr	Dst addr	Seq num	Pkt id
-------	------	-----------	---------	----------	----------	-------	-----	----------	----------	---------	--------

- **Event** Tipo do evento, cada um dos 4 objectos define um evento distinto:
 - EnqT → + (indica que o pacote chegou a fila de espera da ligação)
 - DeqT → - (indica que o pacote saiu da fila de espera da ligação)
 - DrpT → *d* (indica que o pacote foi descartado da fila de espera)
 - RecvT → *r* (indica que o pacote chegou ao nó destino desta ligação)
- **Time** Instante de tempo no qual o evento ocorre;
- **From node** Nó de entrada da ligação na qual ocorre o evento;
- **To Node** Nó de saída da ligação na qual ocorre o evento;
- **Pkt type** Tipo de pacote enviado, por exemplo pode ser: ACK, TCP, CBR, etc...
- **Pkt size** Tamanho do pacote em *bytes*;
- **Flags** O NS apenas utiliza uma das *flags*, a ECN (Explicit Congestion Notification);
- **Fid** Utilizado para diferenciar os fluxos, permite compará-los e/ou visualiza-los com cores diferentes no *NAM*;
- **Src addr** Indica qual o nó e qual a porta utilizada por esse nó para enviar o pacote (node.port). Por exemplo `src_addr = 3.0` indica que o nó origem é o 3 e que o pacote foi enviado pela porta 0;
- **Dst addr** Indica qual o nó de destino e respectiva porta que recebe o pacote;
- **Seq num** Número de série do pacote e é sequencial. Todos os pacotes têm um número sequencial associado, até os pacotes do tipo UDP, embora esse campo não seja utilizado na implementação do protocolo;
- **Pkt id** Identificação do pacote e é única durante a simulação.

Existem na Internet algumas páginas que descrevem os diferentes tipos de formato do ficheiro de registo, destacando a seguinte:

- <http://k-lug.org/~griswold/NS2/ns2-trace-formats.html>

Registo de alteração do valor de uma variável

É possível registar num ficheiro as alterações das variáveis declaradas como `TracedInt` ou `TracedDouble`.

Os agentes TCP têm as suas variáveis de estado declaradas como `TracedInt` ou `TracedDouble`, o que permite com os seguintes comandos registar num ficheiro as alterações de uma ou mais variáveis:

```
set trace_var [open out_var.tr w]
...
$src1 attach $trace_var
$src1 trace ack_
$src1 trace cwnd_
```

É necessário não esquecer de fechar o ficheiro `$trace_var` dentro do procedimento `finish()`.

O ficheiro de saída tem o seguinte formato:

```
0.00000 0 1 1 1 ack_-1
0.00000 0 1 1 1 cwnd_ 1.000
0.83896 0 1 1 1 ack_ 0
0.83896 0 1 1 1 cwnd_ 2.000
0.86760 0 1 1 1 ack_ 1
0.86760 0 1 1 1 cwnd_ 3.000
1.07116 0 1 1 1 ack_ 2
1.07116 0 1 1 1 cwnd_ 4.000
2.07116 0 1 1 1 cwnd_ 1.000
2.23283 0 1 1 1 ack_ 3
2.23283 0 1 1 1 cwnd_ 2.000
2.43559 0 1 1 1 ack_ 4
2.43559 0 1 1 1 cwnd_ 3.000
3.43559 0 1 1 1 cwnd_ 1.000
3.88545 0 1 1 1 ack_ 5
3.88545 0 1 1 1 cwnd_ 2.000
4.06419 0 1 1 1 ack_ 6
4.06419 0 1 1 1 cwnd_ 3.000
```

O comando UNIX `grep` pode ser utilizado para filtrar o ficheiro. Com esse comando é possível separar as variáveis em ficheiros distintos. No caso do ficheiro do registo das variáveis pode-se desejar um ficheiro apenas com a informação da variável `cwnd_`, utilizando o seguinte comando dentro do procedimento `finish()`:

```
exec grep "cwnd_" out_var.tr > out_cwnd_var.tr
```

Obtendo o seguinte resultado:

```
0.00000 0 1 1 1 cwnd_ 1.000
```



```
0.83896 0 1 1 1 cwnd_ 2.000
0.86760 0 1 1 1 cwnd_ 3.000
1.07116 0 1 1 1 cwnd_ 4.000
2.07116 0 1 1 1 cwnd_ 1.000
2.23283 0 1 1 1 cwnd_ 2.000
2.43559 0 1 1 1 cwnd_ 3.000
3.43559 0 1 1 1 cwnd_ 1.000
3.88545 0 1 1 1 cwnd_ 2.000
4.06419 0 1 1 1 cwnd_ 3.000
```

Registo personalizados

Pretende-se por exemplo registar o instante e o valor de `cwnd_` com uma certa periodicidade (em cada 0,5 instantes):

1. Criar o ficheiro para escrita:

```
set trace_cwnd [open trace_cwnd.tr w]
```

2. Criar o procedimento:

```
proc record_trace {} {
    global ns src1 trace_cwnd

    set periodo 0.5
    set now [$ns now]

    set cwnd [$src1 set cwnd_]

    puts $trace_cwnd "$now $cwnd"
    $ns at [expr $now+$periodo] "record_trace"
}
```

3. Agendar o primeiro evento do procedimento:

```
$ns at 0.0 "record_trace"
```

É necessário não esquecer de fechar o ficheiro `$trace_cwnd` dentro do procedimento `finish()`.

Obtendo o seguinte resultado::

```
0 1
0.5 1
1 3
1.5 4
2 4
2.5 3
3 3
3.5 1
4 2
4.5 3
```

D.4. Monitorização da ligação

O objecto utilizado para monitorizar uma fila de espera é o *QueueMonitor*, que por defeito monitoriza com intervalo de 0,1 s (esse valor é parametrizável).

Localização do objecto C++: ...\\ns-2.26\\tools**queue-monitor.cc**

O objecto *QueueMonitor* é associado à ligação que se pretende analisar através do seguinte comando:

```
set qmon [$ns monitor-queue $node0 $node1 ""]
```

O objecto tem alguns contadores estatísticos associados as seguintes variáveis:

- `parrivals_` Número de pacotes que chegaram à fila;
- `barrivals_` Total de *bytes* que entraram na fila;
- `pdepartures_` Número de pacotes que saíram da fila;
- `bdepartures_` Total de *bytes* que saíram da fila;
- `pdrops_` Número de pacotes que a fila descartou;
- `bdrops_` Total de *bytes* que a fila descartou.

E possui variáveis de estado associadas as seguintes variáveis:

- `size_` Total de *bytes* na fila;
- `pkts_` Números de pacotes na fila.

O objecto permite **obter o valor médio das variáveis de estado**. Esse valor é obtido através de um objecto *Integrator* que fornece uma implementação simples do cálculo integral a partir de somas discretas (a variável que fornece esse valor é `sum_`). Dividindo esse valor pelo intervalo de tempo, obtém-se o valor médio de cada uma das variáveis. O comando `reset` inicializa o objecto.

Localização do objecto C++: ...\\ns-2.26\\tools**integrator.cc**

O objecto *Integrator* tem que ser criado para cada variável e é necessário activar essas funcionalidades no objecto *QueueMonitor*:

```
set intpkts [new Integrator]
$qmon set-pkts-integrator $intpkts
set intqmon_pkts [$qmon get-pkts-integrator]
```

```
set intbytes [new Integrator]
$qmon set-bytes-integrator $intbytes
set intqmon_bytes [$qmon get-bytes-integrator]
```

O objecto *QueueMonitor* permite também **calcular o atraso médio dos pacotes na fila de espera**. O valor do atraso de cada pacote é registado num Objecto do tipo *Samples* (criado dentro do Objecto *Integrator*) que permite obter o valor médio dos valores colocados nesse objecto, através do resultado do comando `mean`.

Através desse objecto também é possível obter o valor da variância através do comando `variance` ou utilizando o comando `cnt` contar o número de elementos guardados no objecto. O comando `reset` inicializa o objecto.

É necessário criar na simulação um objecto do tipo *Samples* e activar essas funcionalidades no objecto *QueueMonitor*:

```
set delaysamp [new Samples]
$qmon set-delay-samples $delaysamp
set delays [$qmon get-delay-samples]
```

Analisando a classe C++ `queue-monitor.cc` verifica-se que para além das variáveis descritas o *QueueMonitor* permite obter outros parâmetros de desempenho, como por exemplo calcular o *RTT*.

Estas variáveis podem ser utilizadas para efectuar **uma análise numérica** da fila de espera, lendo os seus valores pretendidos no final da simulação dentro da função `finish {}`:

```
proc finish {} {
...
    set parrivals [$qmon set parrivals_]
    set barrivals [$qmon set barrivals_]
    set pdepartures [$qmon set pdepartures_]
    set bdepartures [$qmon set bdepartures_]
    set pdrops [$qmon set pdrops_]
    set bdrops [$qmon set bdrops_]
    set pkts [$qmon set pkts_]
    set size [$qmon set size_]

    set delaymean [$delays mean]
    set pktsArea [$intqmon_pkts set sum_]
    set pmed [expr 1.0*$pktsArea/$temposim]
    set bytesArea [$intqmon_bytes set sum_]
    set bmed [expr 8.0*$bytesArea/$temposim]
...
    puts "Queue Monitor:"
    puts [format "Arrivals: %7d (pkts) %10d (bytes)"$parrivals $barrivals]
    puts [format "Drops: %7d (pkts) %10d (bytes)"$pdrops $bdrops]
```

```
puts [format "Departures: %7d (pkts) %10d (bytes)" $pdepartures $bdepartures]
puts [format "Current Queue Size: %7d (pkts) %10d (bytes)" $pkts $size]

puts [format "Average Queue Size: %5.2f (pkts) %8.2f (bytes) " $pmed $bmed]
puts [format "Average Delay Time in the Queue : %5.2f (second)" $delaymean]
...
}
```

Ou para **uma análise gráfica** recorrendo-se a um simples procedimento que durante a simulação grava num ou mais ficheiros as variáveis pretendidas para a análise.

```
proc record {} {
    global ns qmon file_stat

    # A variável periodo define o período de tempo entre as leituras
    set periodo 0.5

    set qsize [$qmon set pkts_]
    set now [$ns now]
    puts $file_stat "$now $qsize"

    $ns at [expr $now+$periodo] "record"
}
```

Activando essa funcionalidade num cenário com dois fluxos verifica-se que a análise efectuada não distingue os tráfegos, tratando os fluxos como sendo apenas um.

Para distinguir os fluxos utiliza-se o objecto *QueueMonitor/ED/Flowmon*, que deriva do objecto *QueueMonitor*.

Estão disponíveis as seguintes interfaces OTcl:

- `classifier` Indica os parâmetros que define um fluxo;
- `attach` Associa ao *FlowMonitor* um ficheiro de *trace*;
- `dump` Escreve as variáveis disponíveis no *FlowMonitor* no ficheiro indicado;
- `flows` Indica todos os fluxos cadastrados no *FlowMonitor*.

Um fluxo pode ser classificado, utilizando diferentes campos do pacote, de três formas:

- o `fid` (campo do pacote utilizado para diferenciar os fluxos)
- o `src_addr + dst_addr`
- o `src_addr + dst_addr + fid`

O seguinte comando define como são classificados os fluxos:

```
set flowmon [$ns makeflowmon Fid]
```

É necessário atribuir a cada agente um *fid* distinto de modo a poder distingui-los:

```
...
set src0 [new Agent/UDP]
$src0 set fid 0
...
set src1 [new Agent/TCP]
$src1 set fid 1
...
```

Depois de definido como classificar os fluxos, no exemplo pelo *fid*, é necessário associar o objecto *FlowMonitor* à ligação a monitorizar:

```
set nlink [$ns link $node0 $node1]
$ns attach-fmon $nlink $flowmon
```

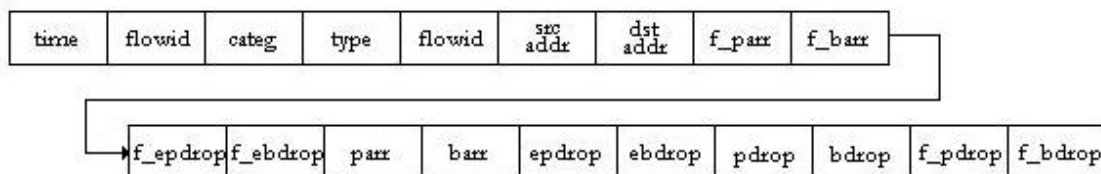
Finalmente criar um ficheiro e associar ao objecto a esse ficheiro no qual o objecto vai escrever em cada execução do comando *dump*.

```
set flow_stat [open flow_stat.stat w]
...
$flowmon attach $flow_stat
...
```

Adicionando ao procedimento *record()* a seguinte linha de comando:

```
proc record {} {
...
$flowmon dump
...
}
```

Obtém-se num ficheiro de trace os parâmetros de desempenho de cada fluxo. O ficheiro tem o seguinte formato:



A informação disponível em cada linha é a seguinte:

- time Instante da leitura das variáveis;
- flowid Valor do *fid* definido para diferenciar os fluxos;
- categ Igual a zero, só é utilizado por razões de compatibilidade com o NS versão 1;
- type Valor lido no cabeçalho comum do pacote;
- flowid Valor do *fid* definido para diferenciar os fluxos;
- src addr Indica qual o nó de origem deste fluxo;

➤ dest_addr	Indica qual o nó de destino deste fluxo;
➤ f_parr	Número de pacotes deste fluxo que chegam a fila;
➤ f_barr	Quantidade de <i>bytes</i> deste fluxo que chegam a fila;
➤ f_epdrop	Número de pacotes que foram descartados sem que a fila esteja cheia (<i>early drop</i>);
➤ f_ebdrop	Quantidade de <i>bytes</i> que foram descartados sem que a fila esteja cheia (<i>early drop</i>);
➤ parr	Número de pacotes que chegam à fila;
➤ barr	Quantidade de <i>bytes</i> que chega a fila;
➤ epdrop	Números de pacotes descartados sem que a fila esteja cheia (<i>early drop</i>);
➤ ebdrop	Quantidade de <i>bytes</i> descartados sem que a fila esteja cheia (<i>early drop</i>);
➤ pdrop	Número de pacotes descartados;
➤ bdrop	Quantidade de <i>bytes</i> descartados;
➤ f_pdrop	Número de pacotes deste fluxo descartados (incluindo os <i>early</i> descartados);
➤ f_bdrop	Quantidade de <i>bytes</i> deste fluxo descartados (incluindo os <i>early</i> descartados);

Exemplo de um ficheiro do *FlowMonitor*:

```

1.000 1 0 0 1 0 1 2 1080 0 0 2 1080 0 0 0 0 0 0
1.500 1 0 0 1 0 1 3 2120 0 0 6 5120 0 0 0 0 0 0
1.500 0 0 29 0 0 1 3 3000 0 0 6 5120 0 0 0 0 0 0
2.000 1 0 0 1 0 1 3 2120 0 0 12 11120 0 0 0 0 0 0
2.000 0 0 29 0 0 1 9 9000 0 0 12 11120 0 0 0 0 0 0
2.500 1 0 0 1 0 1 4 3160 0 0 20 19160 0 0 0 0 0 0
2.500 0 0 29 0 0 1 16 16000 0 0 20 19160 0 0 0 0 0 0
3.000 1 0 0 1 0 1 4 3160 0 0 26 25160 0 0 0 0 0 0
3.000 0 0 29 0 0 1 22 22000 0 0 26 25160 0 0 0 0 0 0
3.500 1 0 0 1 0 1 5 4200 0 0 32 31200 0 0 0 0 0 0
3.500 0 0 29 0 0 1 27 27000 0 0 32 31200 0 0 0 0 0 0
4.000 1 0 0 1 0 1 6 5240 0 0 33 32240 0 0 0 0 0 0
4.000 0 0 29 0 0 1 27 27000 0 0 33 32240 0 0 0 0 0 0
4.500 1 0 0 1 0 1 9 8360 0 0 39 38360 0 0 0 0 0 0
4.500 0 0 29 0 0 1 30 30000 0 0 39 38360 0 0 0 0 0 0

```

A criação de um objecto por cada fluxo depende de qual a designação escolhida para definir um fluxo. No caso de ter sido apenas o *fid*, só é necessário afectar o terceiro campo (a **negrito**) da função `lookup()` para diferenciar os fluxos. o exemplo em baixo é para o fluxo com *fid* = 1.

```

set fcla [$flowmon classifier]
set flow2 [$fcla lookup auto 0 0 1]

```

Para **uma análise numérica** do fluxo com *fid*=0:

```

proc finish {} {
...
set fcla [$flowmon classifier]
set flow0 [$fcla lookup auto 0 0 0]

set parrivals_f0 [$flow0 set parrivals_]
set barrivals_f0 [$flow0 set barrivals_]
set pdepartures_f0 [$flow0 set pdepartures_]
set bdepartures_f0 [$flow0 set bdepartures_]

```

```
set pdrops_f0 [$flow0 set pdrops_]
set bdrops_f0 [$flow0 set bdrops_]
set pkts_f0 [$flow0 set pkts_]
set size_f0 [$flow0 set size_]

puts "Queue Flow Monitor:"
puts "Flow 0:"
puts [format "Arrivals: %7d (pkts) %10d (bytes)" $parrivals_f0
$barrivals_f0]
puts [format "Drops: %7d (pkts) %10d (bytes)" $pdrops_f0 $bdrops_f0]
puts [format "Departures: %7d (pkts) %10d (bytes)" $pdepartures_f0
$bdepartures_f0]
puts [format "Current Queue Size Ocuped: %7d (pkts) %10d (bytes)" $pkts_f0
$size_f0]
...
}
```

Para **uma análise gráfica**, comparando o tamanho da fila de espera ocupada em pacotes por cada fluxo é necessário adicionar os seguintes linhas de código aos procedimentos `Record()` , `finish()` e previamente ter criado os ficheiros para escrita:

```
set file_stat_f0 [open file_f0.stat w]
set file_stat_f1 [open file_f1.stat w]

proc record {} {
...
set fcla [$flowmon classifier]
set flow0 [$fcla lookup auto 0 0 0]
if { $flow0 != "" } {
    set qsize_f0 [$flow0 set pkts_]
    puts $file_stat_f0 "$now $qsize_f0"
}
set flow1 [$fcla lookup auto 0 0 1]
if { $flow1 != "" } {
    set qsize_f1 [$flow1 set pkts_]
    puts $file_stat_f1 "$now $qsize_f1"
}

$ns at [expr $now+$periodo] "record"
}
```

D.5. Ferramentas de Visualização

NAM

O *NAM* é baseado num ficheiro de recolha da actividade da rede, criado e activado com os seguintes comandos:

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

Para correr só é necessário dentro do procedimento `finish()` do *script*, fechar o ficheiro e executar o comando `nam`:

```
close $nf
...
exec nam out.nam &
```

Configuração de alguns parâmetros da aplicação *NAM*:

- Orientar a localização dos nós:

```
$ns duplex-link-op $node0 $node1 orient opção
```

onde **opção** pode ser:

- Right
- Left
- Right-down
- Left-down
- Right-up
- Left-up

- Representar a fila de espera:

```
$ns duplex-link-op $node0 $node1 queuePos valor
```

Onde valor pode ser:

- 0 – Horizontal (sentido para a direita)
- 0,5 – Vertical (sentido para cima)
- 1 – Horizontal (sentido para a esquerda)
- 1,5 – Vertical (sentido para baixo)

- Distinguir os fluxos com cores associando uma cor ao número fid:

```
$ns color 0 Blue
$ns color 1 Red
```

Várias cores disponíveis: red, blue, green, purple, yellow...

- Colorir os nós:

```
$node0 color green
```

- Alterar o formato dos nós que por defeito são redondos:

```
$node0 shape box
```


Podendo também ser hexagon ou circle.

- o Colorir as ligações:

```
$ns duplex-link-op $node0 $node1 color violet
```

- o Adicionar etiquetas. A etiqueta pode aparecer num determinado instante:

```
$ns at 1.2 "$node0 label \"Começa a tx\""
```

ou colocar a etiqueta por exemplo numa determinada ligação:

```
$ns duplex-link-op $node0 $node1 label "Interligação Partilhada"
```

- o Adicionar comentários de texto na parte inferior do *NAM*:

```
$ns at 1 "$ns trace-annotate \"TCP começa a Transmitir\""
```

Xgraph

O comando utilizado para produzir os gráficos é o seguinte:

```
exec xgraph ficheiro1 ficheiro2 ...
```

Configuração de alguns parâmetros do programa Xgraph:

- o Colocar um título ao gráfico:

```
-t "Ocupacao da Fila de Espera"
```

- o Dimensionar o gráfico:

```
-geometry xsize x ysize
```

- o Colocar legendas nos eixos:

```
-x "Segundos" -y "Pacotes"
```

- o Retirar a grelha:

```
-tk
```

- o Alterar os eixos x e/ou y para a escala algorítmica:

```
-lnx -lny
```

- o Marcar os dados com pontos:

- p (com pequenos pontos) -P (com grandes pontos)
- o Desenhar apenas os pontos:
-nl -P
- o Criar automaticamente um ficheiro de postscript:
-device ps -o nome_do_ficheiro.ps

Gnuplot

Para utilizar o Gnuplot dentro do script Tcl é necessário:

- o Criar um ficheiro auxiliar:
set gno [open gno_file.plot w]
- o Escrever no ficheiro os comandos para o Gnuplot:
puts \$gno "plot 'queue_stat.stat', 'file_f0.stat' , 'file_f1.stat'"
- o Fechar o ficheiro:
close \$gno
- o Correr a aplicação Gnuplot dentro do *script*
exec gnuplot -persist gno_file.plot

A opção persist é para o Gnuplot não fechar a janela do gráfico quando esse programa é encerrado no final da simulação.

Configuração de alguns parâmetros do programa Xgraph:

- o Desenhar o gráfico com linhas:
plot "nome_ficheiro" with lines (abreviatura w l)

É possível escolher a cor a linha colocando a seguir a lines o número da cor (disponíveis 1 a 31 cores).

- o Desenhar o gráfico com pontos:

```
plot "nome_ficheiro" with points (abreviatura w p)
```

É possível escolher a forma do ponto colocando a seguir a points o número do objecto (disponíveis 1 a 31 cores).

- Desenhar o gráfico com linhas com pontos:

```
plot "nome_ficheiro" with linespoints (abreviatura w lp)
```

É possível escolher a forma do ponto colocando, a seguir a linespoints, o número do objecto (disponíveis 1 a 31 cores).

- Colocar uma grelha no gráfico:

```
set grid
```

- Escolher a localização das legendas

```
set key opção
```

onde opção:

- Left
- Right
- Top
- Bottom
- Outside, ao lado do gráfico;
- Below, por debaixo do gráfico.

Pode-se combinar as opções, por exemplo: `set key left top`.

- Colocar as legendas dentro de uma caixa com título:

```
set key left title "Legenda" box 3
```

É possível escolher a cor da caixa colocando, a seguir a box, o número da cor (disponíveis 1 a 31 cores).

- Escrever um título para o gráfico:

```
set title "Ocupacao da fila de Espera"
```

- Colocar etiquetas nos eixos:

```
set xlabel "Segundos"
set ylabel "Pacotes"
```

- Colocar uma etiqueta a cada ficheiro:

```
plot "nome_fich1" t "Fich1", "nome_fich2" t "Fich2"
```

- Escolher quais as colunas do ficheiro a utilizarem:

```
plot "nome_ficheiro1" using 1:3 (Utiliza as colunas 1 e 3 do ficheiro)
```

- Gravar o gráfico num ficheiro:

```
set term png
set output "ocupacao.png"
```

É possível gravar esse ficheiro em numero formatos: *gif*, *corel*, *postscript*, *tgif*, *pdf*, *jpeg*, *hpgl*...

E. Código Tcl do *script* exemplo

```
#Iniciar a variável temposim
#set temposim 10

set SINTAX "A syntax é: ns meu_script.tcl <tempo de simulação>"
if {$argc != 1} {puts $SINTAX; exit 0}

set temposim [lindex $argv 0]

#Criar Objecto Simulator
set ns [new Simulator]

#Associar cores ao FID
$ns color 0 Blue
$ns color 1 Red

#
# Ficheiros
#
set queue_stat [open queue_stat.stat w]
set flow_stat [open flow_stat.stat w]

set file_stat_f0 [open file_f0.stat w]
set file_stat_f1 [open file_f1.stat w]

set gno [open gno_file.plot w]

set nf [open out.nam w]
$ns namtrace-all $nf

#
# Criar a Topologia
#

#Criar os nós
set node0 [$ns node]
set node1 [$ns node]

#Criar a interligação
$ns duplex-link $node0 $node1 64kb 20ms DropTail

#Configurações do NAM
$node1 shape box
$ns duplex-link-op $node0 $node1 orient right
$ns duplex-link-op $node0 $node1 queuePos 0.5

#
# Criar um objecto queuemonitor para analisar a fila de espera
#
set qmon [$ns monitor-queue $node0 $node1 ""]

#
# Criar um objecto flowmonitor para analisar os fluxos da fila de espera
#
set flowmon [$ns makeflowmon Fid]

# Indicar qual a interligação a monitorizar
set nlink [$ns link $node0 $node1]
$ns attach-fmon $nlink $flowmon

#Associar ao objecto flowmonitor um ficheiro
$flowmon attach $flow_stat

#
# Activar os integrator do objecto queuemonitor
#
set intpkts [new Integrator]
$qmon set-pkts-integrator $intpkts
```

```
set intqmon_pkts [$qmon get-pkts-integrator]

set intbytes [new Integrator]
$qmon set-bytes-integrator $intbytes
set intqmon_bytes [$qmon get-bytes-integrator]

#
# Activar o delay-samples do objecto queuemonitor
#
set delaysamp [new Samples]
$qmon set-delay-samples $delaysamp
set delays [$qmon get-delay-samples]

#
# Criar os Agentes
#

# Criar agente UDP origem e associá-lo ao nó 0
set src0 [new Agent/UDP]
$src0 set fid_ 0
$ns attach-agent $node0 $src0

# Criar agente UDP destino e associá-lo ao nó 1
set sink0 [new Agent/NodeMonitor]
$ns attach-agent $node1 $sink0

# Conectar os agentes UDP
$ns connect $src0 $sink0

# Criar agente TCP origem e associá-lo ao nó 0
set src1 [new Agent/TCP]
$src1 set fid_ 1
$ns attach-agent $node0 $src1

# Criar agente TCP destino e associá-lo ao nó 1
set sink1 [new Agent/TCPSink]
$ns attach-agent $node1 $sink1

# Conectar os agentes TCP
$ns connect $src1 $sink1

#
# Criar as Fontes de Tráfego
#

#Criar a fonte de tráfego do agente UDP
set traffic0 [new Application/Traffic/Exponential]
$traffic0 set packetSize_ 1000
$traffic0 set burst_time_ 500ms
$traffic0 set idle_time_ 500ms
$traffic0 set rate_ 100k

#Associar a fonte ao agente UDP
$traffic0 attach-agent $src0

#Criar a fonte de tráfego do agente TCP
set traffic1 [new Application/Telnet]

#Associar a fonte ao agente TCP
$traffic1 attach-agent $src1

#
# Procedimentos
#

# Procedimento record
proc record {} {
    global ns qmon queue_stat flowmon file_stat_f0 file_stat_f1

    # A variável time define o período de tempo entre as leituras
    set periodo 0.5

    set qsize [$qmon set pkts_]
```

```

set now [$ns now]
puts $queue_stat "$now $qsize"

$flowmon dump

        set fcla [$flowmon classifier]
set flow0 [$fcla lookup auto 0 0 0]
if { $flow0 != "" } {
        set qsize_f0 [$flow0 set pkts_]
        puts $file_stat_f0 "$now $qsize_f0"
}
set flow1 [$fcla lookup auto 0 0 1]
if { $flow1 != "" } {
        set qsize_f1 [$flow1 set pkts_]
        puts $file_stat_f1 "$now $qsize_f1"
}
$ns at [expr $now+$periodo] "record"
}

# Procedimento finish
proc finish {} {
    global temposim sink0 src1 tr ns queue_stat flow_stat file_stat_f0 global
    file_stat_f1 nf
    global qmon intqmon_pkts intqmon_bytes delays flowmon gno

#Fechar ficheiros abertos
    close $nf
    close $queue_stat
    close $flow_stat
    close $file_stat_f0
    close $file_stat_f1

#Analisar a fila de espera
    set parrivals [$qmon set parrivals_]
    set barrivals [$qmon set barrivals_]
    set pdepartures [$qmon set pdepartures_]
    set bdepartures [$qmon set bdepartures_]
    set pdrops [$qmon set pdrops_]
    set bdrops [$qmon set bdrops_]
    set pkts [$qmon set pkts_]
    set size [$qmon set size_]

    set delaymean [$delays mean]
    set pktsArea [$intqmon_pkts set sum_]
    set pmed [expr 1.0*$pktsArea/$temposim]
    set bytesArea [$intqmon_bytes set sum_]
    set bmed [expr 8.0*$bytesArea/$temposim]

    set fcla [$flowmon classifier]

#Analisar o fluxo com fid = 0
    set flow0 [$fcla lookup auto 0 0 0]

    set parrivals_f0 [$flow0 set parrivals_]
    set barrivals_f0 [$flow0 set barrivals_]
    set pdepartures_f0 [$flow0 set pdepartures_]
    set bdepartures_f0 [$flow0 set bdepartures_]
    set pdrops_f0 [$flow0 set pdrops_]
    set bdrops_f0 [$flow0 set bdrops_]
    set pkts_f0 [$flow0 set pkts_]
    set size_f0 [$flow0 set size_]

#Analisar o fluxo com fid = 1
    set flow1 [$fcla lookup auto 0 0 1]

    set parrivals_f1 [$flow1 set parrivals_]
    set barrivals_f1 [$flow1 set barrivals_]
    set pdepartures_f1 [$flow1 set pdepartures_]
    set bdepartures_f1 [$flow1 set bdepartures_]
    set pdrops_f1 [$flow1 set pdrops_]
    set bdrops_f1 [$flow1 set bdrops_]
    set pkts_f1 [$flow1 set pkts_]
    set size_f1 [$flow1 set size_]

```



```
#Apresentar resultados
puts "Queue Monitor:"
puts [format "Arrivals: %7d (pkts) %10d (bytes)" $parrivals $barrivals]
puts [format "Drops: %7d (pkts) %10d (bytes)" $pdrops $bdrops]
puts [format "Departures: %7d (pkts) %10d (bytes)" $pdepartures $bdepartures]
puts [format "Current Queue Size: %7d (pkts) %10d (bytes)" $pkts $size]

puts [format "Average Queue Size: %5.2f (pkts) %8.2f (bytes)" $pmed $bmed]
puts [format "Average Delay Time in the Queue : %5.2f (second)" $delaymean]
puts ""

puts "Queue Flow Monitor:"
puts "Flow 0:"
puts [format "Arrivals: %7d (pkts) %10d (bytes)" $parrivals_f0 $barrivals_f0]
puts [format "Drops: %7d (pkts) %10d (bytes)" $pdrops_f0 $bdrops_f0]
puts [format "Departures: %7d (pkts) %10d (bytes)" $pdepartures_f0 $bdepartures_f0]
puts [format "Current Queue Size Ocuped: %7d (pkts) %10d (bytes)" $pkts_f0 $size_f0]
puts ""
puts "Flow 1:"
puts [format "Arrivals: %7d (pkts) %10d (bytes)" $parrivals_f1 $barrivals_f1]
puts [format "Drops: %7d (pkts) %10d (bytes)" $pdrops_f1 $bdrops_f1]
puts [format "Departures: %7d (pkts) %10d (bytes)" $pdepartures_f1 $bdepartures_f1]
puts [format "Current Queue Size Ocuped: %7d (pkts) %10d (bytes)" $pkts_f1 $size_f1]
puts ""

#Ficheiro para GNUPLOT

puts $gno "set term png"
puts $gno "set output 'ex_gnuplot.png'"
puts $gno "set key left top"
puts $gno "set key left title 'Legenda' box"
puts $gno "set grid"
puts $gno "set title 'Ocupacao da Fila de Espera'"
puts $gno "set xlabel 'Segundos'"
puts $gno "set ylabel 'Pacotes'"
puts $gno "plot 'queue_stat.stat' w lp pt 3 t 'Ocupacao total', 'file_f0.stat' w lp
pt 8 t 'Ocupacao do fluxo 0', 'file_f1.stat' w lp pt 6 t 'Ocupacao do fluxo 1'

close $gno

# Ferramentas de visualização
exec nam out.nam &
exec xgraph queue_stat.stat file_f0.stat file_f1.stat -geometry 800x400 -t "Ocupacao
da Fila de Espera" -x "Segundos" -y "Pacotes" &
exec gnuplot -persist gno_file.plot &
exit 0
}

#
# Agendar os eventos
#
$ns at 0.0 "$traffic0 start"
$ns at 0.0 "$traffic1 start"
$ns at $temposim "$traffic0 stop"
$ns at $temposim "$traffic1 stop"

$ns at 1.0 "record"

$ns at $temposim "finish"

#
# Iniciar a Simulação
#
puts "A simulação vai começar"
$ns run
```

F. Sistema M/M/1

F.1. Envio de Pacotes

Um método possível de gerar pacotes para simular um sistema M/M/1 é criar duas variáveis aleatórias, *expo1* e *expo2*, independentes e identicamente distribuídas com distribuição exponencial de média $1/\lambda$ e média *b* (tamanho do pacote), respectivamente.

```
#
# 2 exponential distribution variable
#
set expo1 [new RandomVariable/Exponential]
$expo1 set avg_ [expr 1/$lambda]

set expo2 [new RandomVariable/Exponential]
$expo2 set avg_ [expr $b]
```

O procedimento `sendpacket()` foi desenvolvido em Tcl para enviar pacotes com taxa de chegadas *expo1* e com tamanho de pacote *expo2*.

```
proc sendpacket {} {
    global ns src expo1 expo2 b

    set then [$ns now]

    $ns at [expr $then + [$expo1 value]] "sendpacket"

    set bytes [expr round ([$expo2 value]) + 1]

    # Let the Agent send packet with specific size
    $src send $bytes
}
```

Outra solução é utilizar a nova fonte de tráfego, o código para criar a fonte é o seguinte:

```
#
# Generate random seed for RNG values
#
set my_rng [new RNG]
$my_rng seed next-random
#
# Traffic Source
#
set traffic [new Application/Traffic/Exp_Exp]
$traffic set packetSize_ $b
$traffic set intervalTime_ [expr 1000*1/$lambda]ms
$traffic use-rng $my_rng
```

A fonte é configurada para gerar pacotes com tamanho médio **b** e com o valor médio de saída entre pacotes igual a **1/lamdba**.

Variáveis aleatórias

O objecto **my_rng** é utilizado para alterar o seed inicial do gerador de variáveis aleatórias cada vez que a simulação é executada.

A criação do novo objecto que gera números é efectuada através da seguinte linha:

```
set rng [new RNG]
```

Pode-se definir qual a seed inicial através dos seguintes comandos:

```
$rng next-random    Devolve o próximo número aleatório;  
$rng uniform a b    Devolve um número uniformemente distribuído entre [a, b];  
$rng integer k      Devolve um inteiro uniformemente distribuído entre [0, (k-1)];  
$rng exponential    Devolve um n.º de uma distribuição exponencial com média 1.
```

Deste modo em sucessivas repetições, a sequência da geração de números aleatórios não será a mesma.

Depois de criada a fonte é necessário associar essa fonte ao agente:

```
$traffic attach-agent $src
```

É necessário aumentar o tamanho máximo do pacote UDP, porque a nova fonte de tráfego com tamanho médio de pacotes igual a 500 *bytes*, gera pacotes de tamanho superior a 1000 *bytes*, que é o tamanho máximo por defeito. O agente UDP fragmenta um pacote de tamanho 1050 *bytes* em dois pacotes, um de 1000 e outro de 50 *bytes*, o que iria alterar os parâmetros de desempenho da rede.

```
$src set packetSize_ 5000
```

F.2. Cálculo dos parâmetros de desempenho

Os parâmetros de desempenho da fila de espera são obtidos utilizando o objecto *QueueMonitor*:

```
...  
set qmon [$ns monitor-queue $node1 $node2 ""]  
  
set intqm [new Integrator]  
$qmon set-pkts-integrator $intqm  
set intqmon [eval $qmon get-pkts-integrator]  
  
set delaysamp_queue [new Samples]  
$qmon set-delay-samples $delaysamp_queue  
set delay_queue [$qmon get-delay-samples]
```

```
...

proc finish {} {
...
set delaymean_queue [$delay_queue mean]

set pktsInt_queue [$intqmon set sum_]
set pmed_queue [expr 1.0*$pktsInt_queue/$sim_end_time]

...
}
```

O atraso médio no sistema é calculado utilizando o objecto *NodeMonitor*:

```
...
set delaysamp_system [new Samples]
$sink set-delay-samples $delaysamp_system
set delay_system [$sink get-delay-samples]

...

proc finish {} {
...
    set delaymean_system [$delay_system mean]
...
}
```

O objecto *Integrator* é usado para calcular o número médio de pacotes no sistema:

```
set intsystem [new Integrator]
```

O procedimento `record_system()` actualiza em cada instante o valor de pacotes no sistema enviando esse valor para o objecto *Integrator*:

```
proc record_system {} {
global ns traffic sink intsystem qmon
set ns [Simulator instance]
set now [$ns now]
set time 1

set parrivals_queue [$qmon set parrivals_]
set pkts_recv [$sink set npkts_]
set pkts_syst [expr $parrivals_queue - $pkts_recv]

$intsystem newpoint $now $pkts_syst

$ns at [expr $now + $time] "record_system"
}
```

Na função `finish()` é calculada o número médio de pacotes:

```
proc finish {} {
...
set pktsInt_system [$intsystem set sum_]
set pmed_system [expr 1.0*$pktsInt_system/$sim_end_time]
...
}
```

Os resultados obtidos são escritos no ecrã através das seguintes linhas de código dentro da função `finish()`:

```
proc finish {} {  
    ...  
    puts "  ||      Fila de Espera      ||      Sistema "  
    puts "N.º || N.º méd de pkts | Atraso méd  || N.º méd de pkts | Atraso méd"  
    puts "*****"  
    puts [format "%d || %0.3f      | %0.3f      || %0.3f      | %0.3f      " $SimNumber  
    $pmed_queue $delaymean_queue $pmed_system $delaymean_system]  
    exit 0  
}
```

G. Código do Novos Objectos

G.1. Fonte de Tráfego Exp_Exp

Localização: ../ns-2.26/tools/exp-exp.cc

```
#include <stdlib.h>

#include "random.h"
#include "trafgen.h"
#include "ranvar.h"

/* implement an source with exponentially distributed arrival
 * times and exponentially distributed packet size.
 * Parameterized by average arrival time,
 * and average packet size.
 */

class EXP_EXP_Traffic : public TrafficGenerator {
public:
    EXP_EXP_Traffic();
    virtual double next_interval(int&);
    int command(int argc, const char*const* argv);
protected:
    void init();
    double interval_;
    int numpkts_;

    ExponentialRandomVariable intervaltime_;
    ExponentialRandomVariable size_;
};

static class EXP_EXP_TrafficClass : public TclClass {
public:
    EXP_EXP_TrafficClass() : TclClass("Application/Traffic/Exp_Exp") {}
    TclObject* create(int, const char*const*) {
        return (new EXP_EXP_Traffic());
    }
} class_exp_exp_traffic;

// This is a new command that allows us to use
// our own RNG object for random number generation
// when generating application traffic

int EXP_EXP_Traffic::command(int argc, const char*const* argv){
    if(argc==3){
        if (strcmp(argv[1], "use-rng") == 0) {
            intervaltime_.seed((char *)argv[2]);
            size_.seed((char *)argv[2]);
            return (TCL_OK);
        }
    }
    return Application::command(argc,argv);
}

EXP_EXP_Traffic::EXP_EXP_Traffic() : intervaltime_(0.0), size_(0.0)
{
    bind_time("intervalTime_", intervaltime_.avgp());
    bind("packetSize_", size_.avgp());
    bind("numpkts_", &numpkts_);
}

void EXP_EXP_Traffic::init()
{
    numpkts_ = 0;
    interval_ = intervaltime_.value();
}
```

```

        if (agent_)
            agent_>set_pkttype(PT_EXP);
    }

double EXP_EXP_Traffic::next_interval(int& size)
{
    /* start of an idle period, compute idle time */
    double t = intervaltime_.value();
    size = int(size_.value() + .5);
    ++numpkts_;
    return(t);
}

```

G.2. Agente NodeMonitor

Localização: ../ns-2.26/tools/**node-monitor.h**

```

#ifndef ns_node_monitor_h
#define ns_node_monitor_h

#include <tccl.h>

#include "agent.h"
#include "config.h"
#include "packet.h"
#include "ip.h"
#include "rtp.h"
#include "integrator.h"
class NodeMonitor : public Agent {
public:
    NodeMonitor();
    virtual int command(int argc, const char*const* argv);
    virtual void recv(Packet* pkt, Handler*);
protected:
    int npkts_;
    int bytes_;
    int seqno_;
    int nlost_;
    int expected_;
    double last_packet_time_;
    Samples* delaySamp_;
    double maxDelay_;
    double minDelay_;
};

#endif // ns_node_monitor_h

```

Localização: ../ns-2.26/tools/**node-monitor.cc**

```

#include <tccl.h>

#include "agent.h"
#include "config.h"
#include "packet.h"
#include "ip.h"
#include "rtp.h"
#include "node-monitor.h"

static class NodeMonitorClass : public TclClass {
public:
    NodeMonitorClass() : TclClass("Agent/NodeMonitor") {}
    TclObject* create(int, const char*const*) {
        return (new NodeMonitor());
    }
} class_node_mon;

NodeMonitor::NodeMonitor() : Agent(PT_NTYPE), delaySamp_(NULL)

```

```

{
    bytes_ = 0;
    nlost_ = 0;
    npkts_ = 0;
    expected_ = 0;
    last_packet_time_ = 0.;
    maxDelay_ = 0.;
    minDelay_ = 99.;
    seqno_ = 0;
    bind("nlost_", &nlost_);
    bind("expected_", &expected_);
    bind("npkts_", &npkts_);
    bind("bytes_", &bytes_);
    bind("lastPktTime_", &last_packet_time_);
    bind("maxDelay_", &maxDelay_);
    bind("minDelay_", &minDelay_);
}

void NodeMonitor::recv(Packet* pkt, Handler*)
{
    hdr_rtp* p = hdr_rtp::access(pkt);

    seqno_ = p->seqno();
    bytes_ += hdr_cmh::access(pkt)->size();
    ++npkts_;

    double now = Scheduler::instance().clock();

    if (delaySamp_)
        delaySamp_->newPoint(now - hdr_cmh::access(pkt)->txtime());

    /*
     * Update maxDelay & minDelay
     */
    double delay = now - hdr_cmh::access(pkt)->txtime();

    if ( delay > maxDelay_ )
        maxDelay_ = delay;

    if ( delay < minDelay_ )
        minDelay_ = delay;

    /*
     * Check for lost packets
     */
    int loss = seqno_ - expected_;
    if (loss == 0) {
        expected_ = seqno_ + 1;
    }
    else {
        if (loss > 0){
            nlost_ += loss;
            expected_ = seqno_ + 1;
        }
        else {
            nlost_ = nlost_ - 1;
            printf ("Chegou um pacote atrasado");
        }
    }

    last_packet_time_ = now;
    Packet::free(pkt);
}

/*
 * $proc interval $interval
 * $proc size $size
 */
int NodeMonitor::command(int argc, const char*const* argv)
{
    Tcl& tcl = Tcl::instance();

```



```
if (argc == 2) {
    if (strcmp(argv[1], "get-delay-samples") == 0) {
        if (delaySamp_)
            tcl.resultf("%s", delaySamp_>name());
        else
            tcl.resultf("");
        return (TCL_OK);
    }
}
if (argc == 3) {
    if (strcmp(argv[1], "set-delay-samples") == 0) {
        delaySamp_ = (Samples*)
            TclObject::lookup(argv[2]);
        if (delaySamp_ == NULL)
            return (TCL_ERROR);
        return (TCL_OK);
    }
}
return (Agent::command(argc, argv));
}
```

É necessário fazer umas alterações ao objecto Agente UDP (apps/udp.cc), inserindo as linhas a negrito, de modo a colocar o instante da transmissão do pacote no campo escolhido.

```
void UdpAgent::sendmsg(int nbytes, AppData* data, const char* flags)
{
    ...
    double local_time = Scheduler::instance().clock();
    while (n-- > 0) {
        p = allocpkt();
        hdr_cm_n::access(p)->size() = size_;
        hdr_rtp* rh = hdr_rtp::access(p);
        rh->flags() = 0;
        rh->seqno() = ++seqno_;
        hdr_cm_n::access(p)->timestamp() =
            (u_int32_t)(SAMPLERATE*local_time);
        // to calculate end-to-end delay
        hdr_cm_n::access(p)->txtime() = local_time;
        // add "beginning of talkspurt" labels (tcl/ex/test-rcvr.tcl)
        if (flags && (0 == strcmp(flags, "NEW_BURST")))
            rh->flags() |= RTP_M;
        p->setdata(data);
        target->recv(p);
    }
    n = nbytes % size_;
    if (n > 0) {
        p = allocpkt();
        hdr_cm_n::access(p)->size() = n;
        hdr_rtp* rh = hdr_rtp::access(p);
        rh->flags() = 0;
        rh->seqno() = ++seqno_;
        hdr_cm_n::access(p)->timestamp() =
            (u_int32_t)(SAMPLERATE*local_time);
        hdr_cm_n::access(p)->txtime() = local_time;
    }
    ...
}
```

G.3. Adicionar novo objecto ao NS

Depois de criado um novo objecto é necessário integrá-lo no NS de modo a ficar disponível para futuras simulações.

Descrição dos passos a seguir:

1. Colocar o ficheiro no directório específico
(Por exemplo:...\NS-2.26\novos\objecto.cc);
2. Inicializar as variáveis acessíveis no OTcl no ficheiro:
...\ns-2.26\tcl\lib**ns-default.tcl**;
3. Entrar no directório: `cd ...\NS-2.26`
4. Alterar o **makefile**, acrescentando o novo objecto (**novos/objecto.o**) a lista de objectos;
5. Executar o comando `make depend`;
6. Por fim executar o comando `make`.

H. Técnica de descarte - RED

H.1. Alterações ao *script* do Tutorial *NS - by example*

O método utilizado no tutorial para obter o valor instantâneo e médio do tamanho da fila de espera é o seguinte:

```
# Tracing a queue
set redq [[ $ns link $node_(r1) $node_(r2) ] queue]
set tchan_ [open all.q w]
$redq trace curq_
$redq trace ave_
$redq attach $tchan_
```

As variáveis `curq_` e `ave_` não estão disponíveis para o objecto `DropTail`, ou seja para o tipo `DropTail` não é possível calcular esses valores através do mesmo método.

Para o efeito utiliza-se o objecto *Queue Monitor* e um procedimento designado por `record_queue()`, para recolher os dados pretendidos.

```
...
set qmon [ $ns monitor-queue $node_(r1) $node_(r2) "" ]

set intqm [new Integrator]
$qmon set-pkts-integrator $intqm
set intqmon [ $qmon get-pkts-integrator ]
...

proc record_queue {} {
    ...
    set qpks [ $qmon set pkts_ ]
    set qint [ $intqmon set sum_ ]
    set qmean [expr $qint/$now]
    ...
}
```

Com essa alteração as variáveis `qpks` e `qmean` representam o valor instantâneo e médio do tamanho da fila de espera, respectivamente, independentemente do tipo de fila de espera escolhido.

Apenas para facilitar as simulações, o tipo de fila de espera foi colocado como parâmetro de entrada do *script*:

```
...
set queueType [lindex $argv 0]

$ns duplex-link $node_(r1) $node_(r2) 1.5Mb 20ms $queueType
...
```

Para correr a simulação com uma fila de espera do tipo RED utiliza-se a seguinte linha de comando:

```
ns droptail_red.tcl RED
```

Para o tipo Droptail é só substituir o termo RED por DropTail.

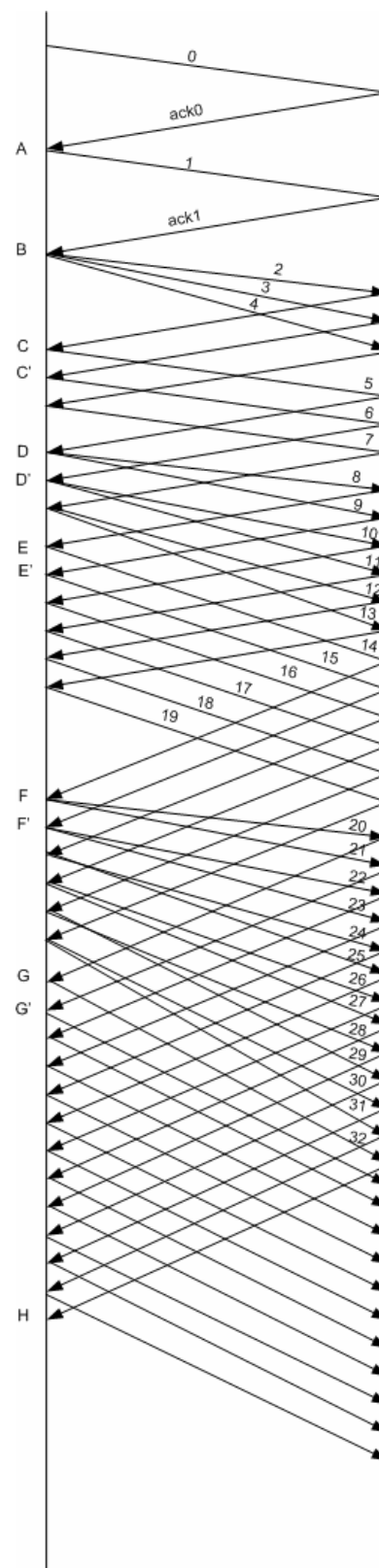
```
ns droptail_red.tcl DropTail
```

A segunda alteração é, em vez de colocar os dados recolhidos num único ficheiro e depois processá-los através de código Awk, os dados são directamente colocados em ficheiros separados.

I. Protocolo de transporte TCP

I.1. *Slow start* do TCP Vegas

Nos resultados obtidos no estudo experimental do TCP Vegas verificou-se a influência do atraso de propagação na passagem da fase *Slow Start* para a fase *Congestion Avoidance*. Na passagem dos 60 para os 70 ms de atraso de propagação o instante da mudança da fase *Slow Start* inicial para a fase *Congestion Avoidance* é alterado. A figura ao lado apresenta a troca de pacotes entre o emissor e o receptor, desde o início da transmissão até a janela de congestionamento do emissor ser igual a 12 pacotes, momento em que para os 60 ms a fase *Slow Start* acaba, o que não acontece para os 70 ms.



Analisando passo-a-passo a troca dos pacotes obtém-se o quadro em baixo, as operações em *itálico* são realizadas durante o RTT não.

	60 ms	70 ms
A	<i>Chega ack do 0</i> $RTT = RTT_0 = 0.137504 = \text{baseRTT}$ $v_sumRTT_ e v_cnt_RTT_ = 0$ $rTTLen = 1$ $actual = 7.27 / esperado = 7.27 / delta = 0$ $Marca\ próximo = 1$ Envia 1	$RTT = RTT_0 = 0.157504 = \text{baseRTT}$
B	<i>Chega ack do 1</i> Actualiza janela = 2 $RTT = RTT_1 = 0.137504$ $v_sumRTT_ e v_cnt_RTT_ = 0$ $rttLen = 1$ $actual = 7.27 / esperado = 7.27 / delta = 0$ $delta < 1$ $Marca\ próximo = 2$ $Cwnd = Cwnd + 1 \ (3)$ Envia 2, 3 e 4	
C	<i>Chega ack do 2</i> $RTT_2 = 0.137504$ $RTT = RTT_i / i$ $v_sumRTT_ e v_cnt_RTT_ = 0$ $rTTLen = 3$ $actual = 21.82 / esperado = 21.82 / delta = 0$ $Marca\ próximo = 5$ Envia 5	
C'	<i>Chega ack do 3</i> $RTT_3 = 0.144504$ Envia 6 <i>Chega ack do 4</i> $RTT_4 = 0.151504$ Envia 7	
D	<i>Chega ack do 5</i> $RTT = (RTT_2 + RTT_3 + RTT_4) / 3 = 0.144504$ $v_sumRTT_ e v_cnt_RTT_ = 0$ $rttLen = 3$ $actual = 20.76 / esperado = 21.82 / delta = 0$ $delta < 1$ $Marca\ próximo = 8$ $Cwnd = Cwnd + 1 \ (4)$ $RTT_5 = 0.137504$ Envia 8 e 9	
D'	<i>Chega ack do 6</i> $Cwnd = Cwnd + 1 \ (5)$ $RTT_6 = 0.137504$ Envia 10 e 11 <i>Chega ack do 7</i> $Cwnd = Cwnd + 1 \ (6)$ $RTT_7 = 0.137504$ Envia 12 e 13	
E	<i>Chega ack do 8</i> $RTT_8 = 0.137504$ $RTT = (RTT_5 + RTT_6 + RTT_7) / 3 = 0.137504$ $v_sumRTT_ e v_cnt_RTT_ = 0$ $rTTLen = 6$	

	$actual = 43.64 / esperado = 43.64 / delta = 0$ $Marca\ próximo = 14$ Envia 14	
E ²	Chega ack do 9 $RTT_9 = 0.144504$ Envia 15 Chega ack do 10 $RTT_{10} = 0.144504$ Envia 16 Chega ack do 11 $RTT_{11} = 0.151504$ Envia 17 Chega ack do 12 $RTT_{12} = 0.151504$ Envia 18 Chega ack do 13 $RTT_{13} = 0.158504$ Envia 19	
F	Chega ack do 14 $RTT_{14} = 0.137504$ $RTT = (RTT_8 + RTT_9 + RTT_{10} + RTT_{11} + RTT_{12} + RTT_{13}) / 6 = 0.148004$ $v_sumRTT_ e\ v_cnt_RTT_ = 0$ $rttLen = 6$ $actual = 40.54 / esperado = 43.64 / delta = 0$ $delta < 1$ $Marca\ próximo = 20$ $Cwnd = Cwnd + 1\ (7)$ Envia 20 e 21	
F ²	Chega ack do 15 $Cwnd = Cwnd + 1\ (8)$ $RTT_{15} = 0.137504$ Envia 22 e 23 Chega ack do 16 $Cwnd = Cwnd + 1\ (9)$ $RTT_{16} = 0.137504$ Envia 24 e 25 Chega ack do 17 $Cwnd = Cwnd + 1\ (10)$ $RTT_{17} = 0.137504$ Envia 26 e 27 Chega ack do 18 $Cwnd = Cwnd + 1\ (11)$ $RTT_{18} = 0.137504$ Envia 28 e 29 Chega ack do 19 $Cwnd = Cwnd + 1\ (12)$ $RTT_{19} = 0.137504$ Envia 30 e 31	
G	Chega ack do 20 $RTT_{20} = 0.137504$ $RTT = (RTT_{14} + RTT_{15} + RTT_{16} + RTT_{17} + RTT_{18} + RTT_{19}) / 6 = 0.137504$ $v_sumRTT_ e\ v_cnt_RTT_ = 0$ $rttLen = 12$ $actual = 87.27 / esperado = 87.27 / delta = 0$ $Marca\ próximo = 32$ Envia 32	
G ²	Chega ack do 21 $RTT_{21} = 0.144504$ Envia 33 Chega ack do 22 $RTT_{22} = 0.144504$ Envia 34	

	<p> Chega ack do 23 $RTT_{23} = 0.151504$ Envia 35 Chega ack do 24 $RTT_{24} = 0.151504$ Envia 36 Chega ack do 25 $RTT_{25} = 0.158504$ Envia 37 Chega ack do 26 $RTT_{26} = 0.158504$ Envia 38 Chega ack do 27 $RTT_{27} = 0.165504$ Envia 39 Chega ack do 28 $RTT_{28} = 0.165504$ Envia 40 Chega ack do 29 $RTT_{29} = 0.172504$ Envia 41 Chega ack do 30 $RTT_{30} = 0.172504$ Envia 42 Chega ack do 31 $RTT_{31} = 0.179504$ Envia 43 </p>	
H	<p> Chega ack do 32 $RTT = (RTT_{20} + RTT_{21} + RTT_{22} + RTT_{23} + RTT_{24} + RTT_{25} + RTT_{26} + RTT_{27} + RTT_{28} + RTT_{29} + RTT_{30} + RTT_{31}) / 12 = 0.158504$ $v_sumRTT_ e v_cnt_RTT_ = 0$ $rttLen = 12$ $actual = 75.71 / esperado = 87.27 / delta = 2$ $delta > 1$ FIM SLOW-START Marca próximo = 44 </p>	<p> $RTT = 0.178504$ $actual = 67.23 / esperado = 76.19$ $delta < 1$ SÓ NO PRÓXIMO RTT SIM é que comuta de fase </p>